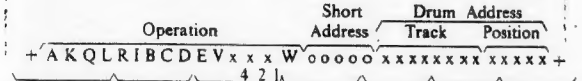
Stantec Zebra

PROGRAM MANUAL

(PART II)

Revised Edition . . . August 1959

PRINTED IN ENGLAND



Standard call-in combinations.

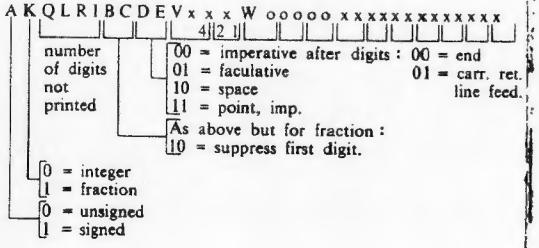
X33P :	Number from tape → A and → B	
X38P :	Change to simple code on instruction (B)	
X39P :	Number from dial → A, number of digits → B	
X40P :	(A) . (B) rounded → A and B	Uses 4,5,6,15.
X40P1 :	(A) . (B) → AB	Uses 4,5,6,15.
X41P :	(A) / (B) rounded → A and B	Uses 4,5,6,7,15.
X42P :	(AB) / (8) → B, remainder → A	
	(A) < (8) (8) undisturbed	Uses 4,5,6,7,8,15

Standard printing.

(A) = number (B) = pattern

X45Pn :	Type
X46Pn :	Type and punch machine code.
X47Pn :	Punch teleprinter code.
X48P :	Punch machine code.
X47P3 :	Set new resident pattern = (B)
n = o :	according to resident pattern.
2 :	according to incident pattern.
4 :	carr. return, line feed, figure shift.
28 :	resident, superpositive.
30 :	incident, superpositive.
31 :	standard integer + 0000000000 spa spa.
32 :	standard fraction +0.000000000 spa spa.

Structure of pattern.



A	P_0
K	P_1
Q	P_2
L	P_3
R	P_4
I	P_5
B	P_6
C	P_7
D	P_8
E	P_9
V	P_{10}
V_4	P_{11}
V_2	P_{12}
V_1	P_{13}
(32)	P_{14}
16	P_{15}
8	P_{16}
4	P_{17}
2	P_{18}
1	P_{19}
4096	P_{20}
2048	P_{21}
1024	P_{22}
512	P_{23}
256	P_{24}
128	P_{25}
64	P_{26}
32	P_{27}
16	P_{28}
8	P_{29}
4	P_{30}
2	P_{31}
1	P_{32}

= 1, output sign; = 0, suppress sign
= 1, convert as fraction; = 0, convert as integer

= i, then output 10 − i decimal digits

P_6 P_7 onwards are taken in pairs, each bit pair having a value k with the following significance

k = 0, output digit imperatively

k = 1, output digit facultatively

k = 2, output a space

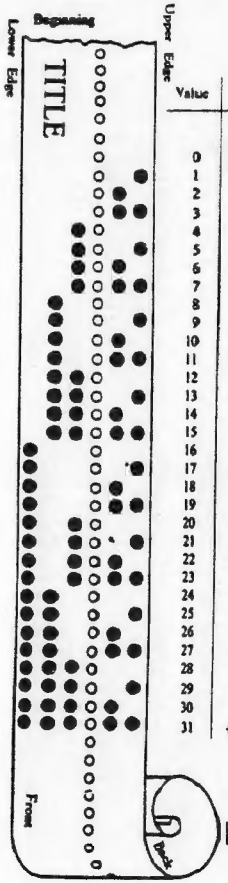
k = 3, output a point

P_6 P_7 = 2 has special meaning on fraction output, suppress most significant digit.

After 10 − i digits have been output
k = 0, leave output program and return to main program

k = 1, output CR/LF
(similarly for P_{21})

Teleprinter Code	
Letters	Figures
E	3
Line Feed	
A	−
Space	
S	'
I	8
U	7
Carr. return	
D	(Tab)
R	4
J	Bell
N	,
F	
C	:
K	(
T	5
Z	+
L	)
W	2
H	
Y	6
P	0
Q	1
O	9
B	?
G	
Figures	
M	.
X	/
V	=
Letters	



Value	Machine Code	
0	0	Blank
1	1	
2	2	
3	3	
4	4	Numerals
5	5	
6	6	
7	7	
8	8	
9	9	
10	K	Short for control
11	Q	Quotient Count
12	.	Decimal Point
13	L	Left Shift
14	R	Right Shift
15	I	Inversion
16	B, H	B Accumulator
17	C, S	Clear
18	D	Store to Drum
19	E	Store to Short Reg.
20	T	Floating Addresses
21	U	Test Switches
22	V	Test Sign, etc.
23	N	Next, Neutralise
24	A	Add
25	X	Jump
26	+	
27	−	
28	Y	Input Indication
29	Z	Execute
30	P	Parameter
31	#	Correction

PART II

Chapter 5 **Logical Description of Stantec-Zebra: Introduction to the Normal Code**

5.1 : The instruction word
5.2 : Operational digits
5.2.1 : The A digit
5.2.2 : The K digit
5.2.3 : Adding jump
5.2.4 : Double jump
5.2.5 : Double addition
5.2.6 : Jumping addition
5.2.7 : The D and E digits
5.3 : The arithmetic unit
5.4 : Further operational digits
5.4.1 : The B Digit
5.4.2 : The Q digit
5.4.3 : The L digit
5.4.4 : The R digit
5.4.5 : The I digit
5.4.6 : The C digit
5.5 : The V, V1, V2, V4 digits
5.5.1 : The U1–U7 combinations
5.5.2 : V1—Test A negative
5.5.3 : V2—Test B negative
5.5.4 : V3—Test A zero
5.5.5 : V4—Test least significant digit of B
5.5.6 : V5, V6, V7
5.5.7 : The V digit
5.6 : The opening symbols: A and X
5.7 : The W digit

Chapter 6 **The Control Unit**

6.1 : The 'C' and 'E' control registers
6.2 : The test and transfer box
6.3 : W = 1
6.4 : The action of X and A instructions: the 'D' control register
6.5 : 'Failing' test instruction: AW
6.6 : Connection between 'D' and register 4
6.7 : Calling in a subroutine
6.8 : Repeated instructions
6.9 : Some simple program examples
6.9.1 : N and NKK
6.9.2 : The L and R digits
6.9.3 : The combination LR
6.9.4 : Double transport
6.9.5 : A note on the I digit
6.10 : Pre-instructions

Chapter 7 **Zebra Programming (1)**

7.1 : The dynamic stop
7.2 : Repeated instructions: multiple shifting
7.3 : Double length addition and subtraction
7.3.1 : Double length addition
7.3.2 : Double length subtraction
7.3.3 : The carry trap: normal action
7.3.4 : The carry trap: exceptional cases

Chapter 7—cont.
7.4 : Double length multiplication: use of register 15
7.4.1 : The action of the multiplication program
7.4.2 : The complete multiplication program
7.4.3 : Instruction analysis: unrounded product
7.4.4 : Rounded product: the V digit
7.5 : Relative drum locations

Chapter 8 **Zebra Programming (2)**
8.1 : The XD facility
8.2 : Division: short process
8.3 : The normal division program
8.4 : Block transport
8.5 : Repeating blocks of instructions
8.6 : Short multiplication
8.6.1 : Multiplication by small fraction
8.6.2 : Multiplication by small integer
8.7 : Conjugation or logical product

Chapter 9 **Zebra Programming (3)**
9.1 : Input instructions: use of registers 26–31
9.2 : Output instructions
9.2.1 : Punch: E26–E31
9.2.2 : Teleprinter: Register 25

Chapter 10 **Input Programs**
10.1 : The pre-input program
10.2 : The short input program
10.2.1 : Location 32
10.2.2 : Character >1
10.2.3 : Binary form
10.2.4 : Parameter indication digit: Register 9
10.2.5 : Input indication digit: Register 11
10.3 : Vertical ladder of the normal input program

Chapter 11 **The Normal Input Program (1)**
11.1 : Construction of instruction part
11.1.1 : Entry with N or X
11.1.2 : Entry with A
11.1.3 : After Y
11.1.4 : After Z
11.2 : Word assembly: Horizontal ladder
11.3 : The special symbols U and V
11.4 : The parameter facility P
11.4.1 : Subroutine call-in
11.4.2 : Mechanism of P facility
11.5 : Cumulative parameters
11.6 : Accumulative parameters
11.7 : Some parameter conventions

Chapter 12 **The Normal Input Program (2)**
12.1 : The T facilities
12.1.1 : T followed by digit
12.1.2 : TA and TX facility
12.1.3 : TP facility: Floating addresses
12.1.4 : TV facility
12.1.5 : TE facility
12.1.6 : TD facility
12.2 : Input of numbers
12.3 : Correction facility

Chapter 12—cont.

12.4 : Subroutine for taking in numbers: X33P
12.5 : Subroutine conventions
12.6 : The telephone dial program: X39P

Chapter 13 **Output Programs**

13.1 : Complete output program
13.1.1 : Layout pattern
13.1.2 : Some samples of output patterns
13.1.3 : Signed double length fractions
13.1.4 : Pattern for normal integer
13.1.5 : Pattern for normal fraction
13.2 : Building blocks
13.2.1 : Building block program for output of digit and sign

Chapter 14 **Program Conventions**

14.1 : Registers
14.2 : Convention regarding subroutines
14.3 : "Dead" programs: Drum storage allocation

Chapter 15 **The Control Panels**

15.1 : The machine control panel
15.1.1 : Cathode ray tube and display selection
15.1.2 : Efficiency meter
15.1.3 : The parity key
15.1.4 : The 16 block keys and the store locking key
15.1.5 : The "instruction word" keys
15.1.6 : Clear key
15.1.7 : Start key
15.1.8 : Conditional stop (c.s.) key; step key; stop key; manual key
15.1.9 : The U1–U6 keys
15.2 : The desk control panel
15.2.1 : Parity light
15.2.2 : Telephone dial
15.2.3 : The loudspeaker
15.2.4 : The emergency button

PART II

LOGICAL DESCRIPTION OF STANTEC-ZEBRA: INTRODUCTION TO THE NORMAL CODE

Chapter 5

5.1 The Instruction Word.

In computers a *word* is generally regarded as a unit of information. In Zebra it can be defined as being a block of 33 binary digits representing either a number or an instruction.

The *number word* in the machine therefore consists of 33 binary digits, designated 0–32 from the most to the least significant, and the first digit is the sign digit.

The *instruction word* also has 33 binary digits, and its construction is one of the characteristics of Zebra.

FIG. 5·1

It can be seen that the instruction word is in three parts :

(1) *The Drum Address*, or *Main Store Address*, consists of 13 binary digits of which the 5 least significant determine the drum position, i.e. its orientation when the instruction is executed, and the more significant 8 digits determine the track of the drum to be selected. These 8 digits are decoded into one of 256 combinations to select one track. The way in which the correct position round the track is found will be clear when the control unit has been described. (See Chapter 6.)

(2) The 5 digits forming the register addresses are decoded to give 32 possible selections. There are twelve true registers, each having a word length of 33 binary digits : the other addresses select accumulators, useful constants, zero, the most significant 1, and the least significant 1, and some are used to operate input and output mechanisms (see 9.1). These latter addresses refer to what are called pseudo-registers.

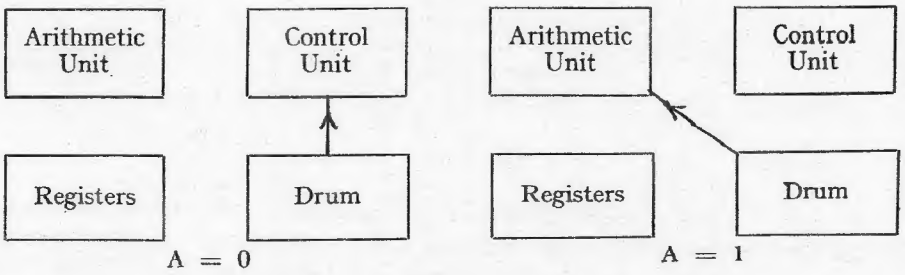
(3) The unusual characteristic of the Zebra instruction word is the size of the function part and the way in which it is used. There are 15 binary digits in this function part and most of these are not decoded at all, as is the usual practice. For when the instruction is staticised in the control unit, each function digit, being a 0 or a 1, operates a switch and causes an elementary operation within the computer. These are called operational digits and are completely independent and individual.

5.2 Operational Digits.

The operational digits are represented by letters : learning the normal code therefore only necessitates the memorising of the functions represented by each of the 15 letters. Let us now consider the action of some of these digits.

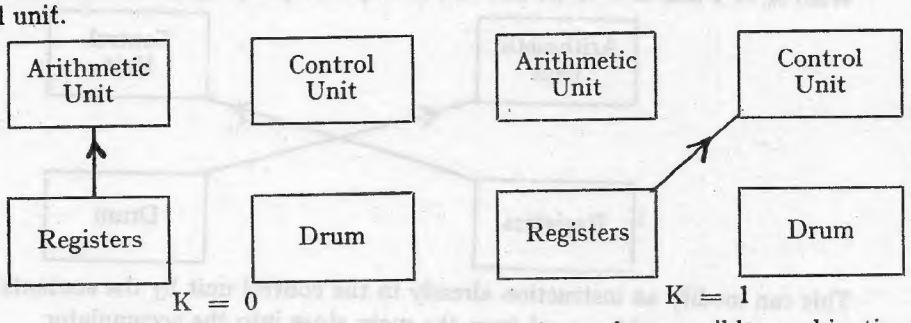
5.2.1 The A Digit.

The most significant digit in the word is the A digit, and this can be 0 or 1. When A = 0 the drum is connected to the control unit. When A = 1 the drum is connected to the arithmetic unit.



5.2.2 **The K Digit.**

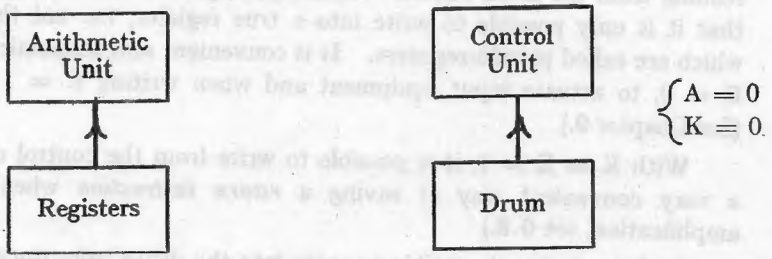
Similarly, the next digit K controls the connection between the registers and the arithmetic or control unit.



The combined action of the A and K digits give rise to four possible combinations which become four specific types of operation.

5.2.3 **Adding Jump.**

When A = 0 and K = 0 and '*adding jump*' is executed.

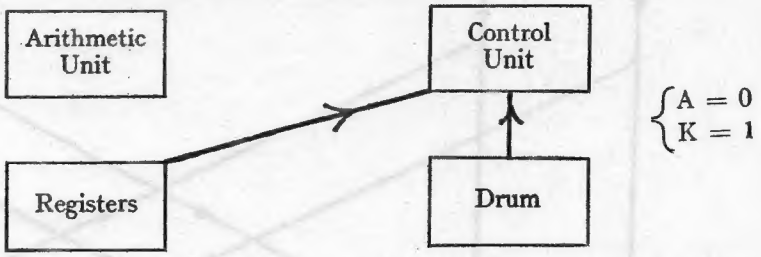


In the adding jump the word specified by the drum address comes from the drum into the control unit. The registers are connected to the arithmetic unit. Thus, although an instruction is being extracted, at the same time arithmetic is being done between the registers and the arithmetic unit.

In this state the machine is behaving as a 1 + 1 address machine.

5.2.4 **Double Jump.**

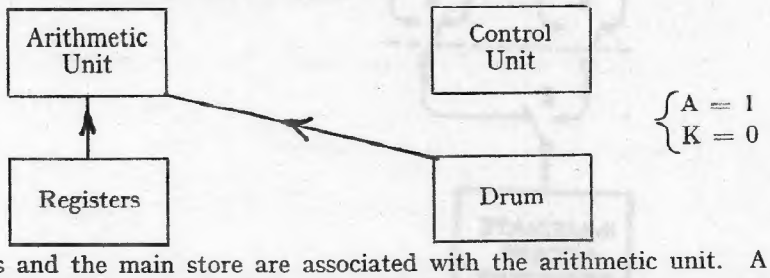
When A = 0 and K = 1 a '*double jump*' is executed.



Here, words from any selected register and a main store location are added to form a new instruction. This is equivalent to modifying the instruction as it stands in the main store by the amount specified in the register. So any register can be used as a '*B' register* or 'order modification' register.

5.2.5 **Double Addition.**

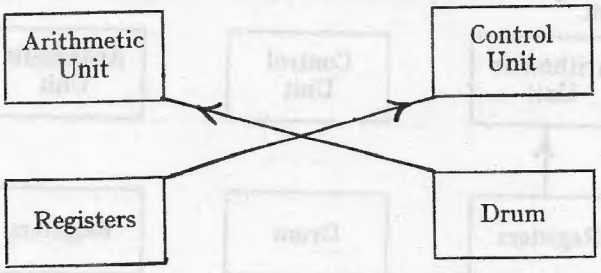
When A = 1 and K = 0 '*double addition*' occurs.



Both the registers and the main store are associated with the arithmetic unit. A word from each can be added and their sum added into the accumulator. The machine is now behaving as a two-address machine with both addresses specifying locations to be used for arithmetic.

5.2.6 Jumping Addition.

With A = 1 and K = 1 we have an example of *'jumping addition'*.



This can modify an instruction already in the control unit by the contents of a register and at the same time can add a word from the main store into the accumulator.

5.2.7 The D and E Digits.

So far it has been assumed that both D and E = 0. When D = 1, writing into instead of reading from the drum occurs. Similarly, with E = 1, writing into the register occurs. Note that it is only possible to write into a true register, i.e. not those containing constants, etc., which are called pseudo-registers. It is convenient and economic to use addresses, when reading E = 0, to actuate input equipment and when writing E = 1 to actuate output equipment. (See Chapter 9.)

With K = E = 1, it is possible to write from the control unit into a register. This gives a very convenient way of saving a *return instruction* when entering a subroutine. (For amplification, see 6.8.)

In Zebra it is not possible to write into the drum from the control unit.

THE ACTION OF A,K,D AND E DIGITS

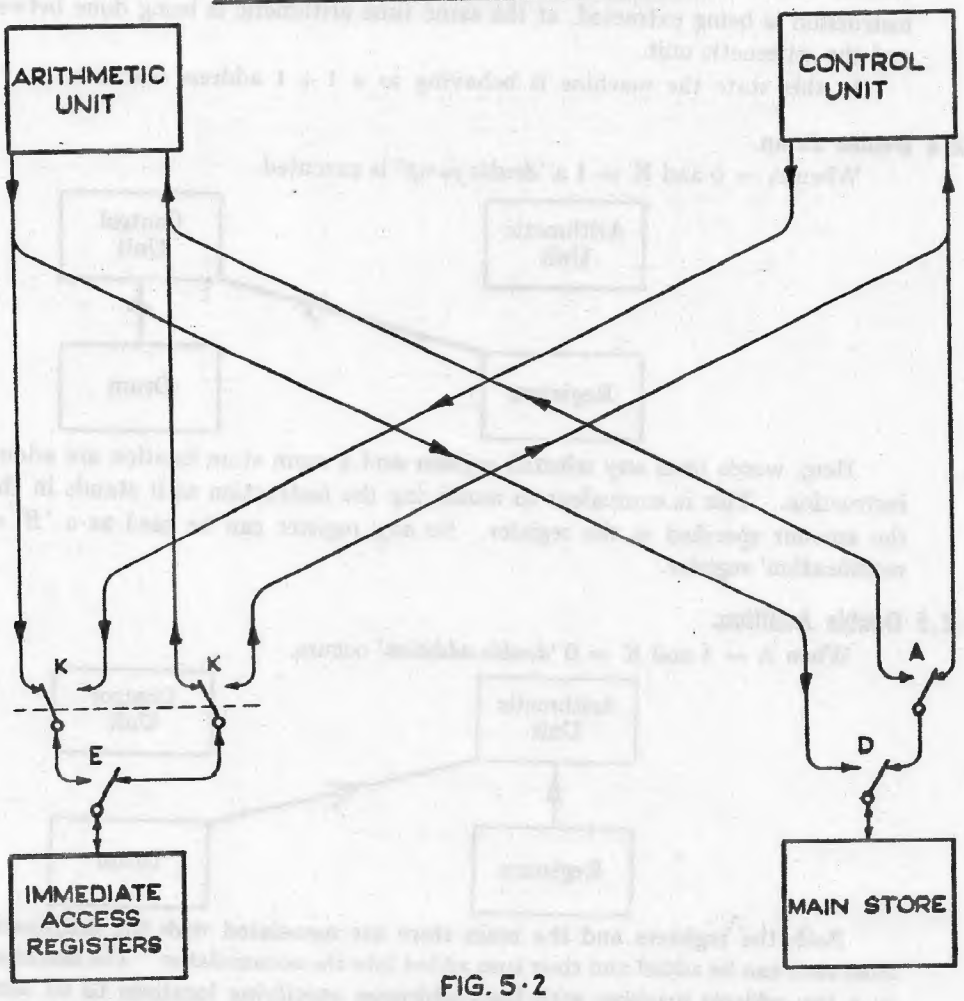


FIG. 5·2

5.3 The Arithmetic Unit.

In the arithmetic unit there are two accumulators known as A and B. Associated with each is a preadder in which the words coming from the registers or drum are first added together before their sum is added into, or subtracted from, the appropriate accumulator. The appropriate accumulator is specified by the B digit.

DIAGRAM OF ARITHMETIC UNIT

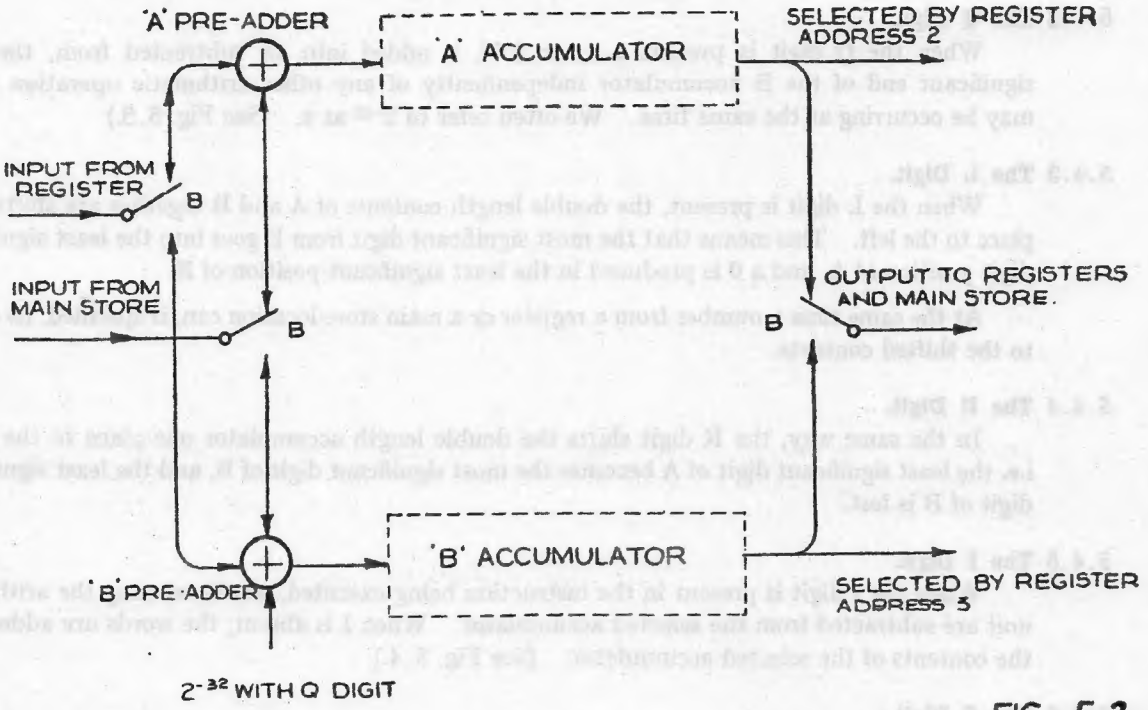


FIG. 5.3.

DIAGRAM OF ACCUMULATORS

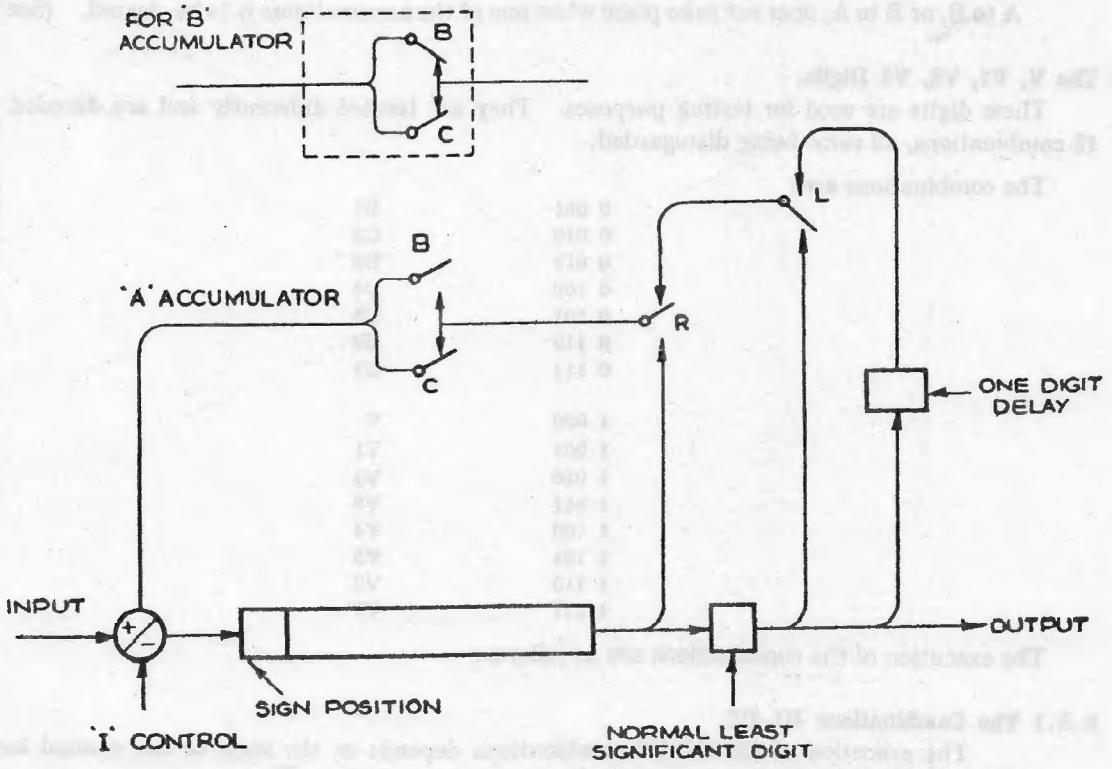


FIG. 5.4.

5.4 Further Operational Digits.

The following digits control the operation of the arithmetic unit.

5.4.1 The B Digit.

When B = 0, the A accumulator is specified; when B = 1 the B accumulator is specified. The inputs to and outputs from the accumulators are controlled by the B digit. The outputs of A and B can be read as if they were registers 2 and 3, respectively. (See Fig. 5.3.)

5.4.2 The Q Digit.

When the Q digit is present a 1, or 2^{-32}, is added into, or subtracted from, the least significant end of the B accumulator independently of any other arithmetic operation which may be occurring at the same time. We often refer to 2^{-32} as ε. (See Fig. 5.3.)

5.4.3 The L Digit.

When the L digit is present, the double length contents of A and B together are shifted one place to the left. This means that the most significant digit from B goes into the least significant digit position of A, and a 0 is produced in the least significant position of B.

At the same time a number from a register or a main store location can, if specified, be added to the shifted contents.

5.4.4 The R Digit.

In the same way, the R digit shifts the double length accumulator one place to the right, i.e. the least significant digit of A becomes the most significant digit of B, and the least significant digit of B is lost.

5.4.5 The I Digit.

When the I digit is present in the instruction being executed, words entering the arithmetic unit are subtracted from the selected accumulator. When I is absent, the words are added into the contents of the selected accumulator. (See Fig. 5.4.)

5.4.6 The C Digit.

When the C digit is present, the accumulator specified by the B digit is cleared. If the contents of the specified accumulator are to be copied into a register or main store location this takes place before the accumulator is cleared. In shifting operations, the shifting from A to B, or B to A, does not take place when one of the accumulators is to be cleared. (See 6.9.2.)

5.5 The V, V1, V2, V4 Digits.

These digits are used for testing purposes. They are treated differently and are decoded to give 15 combinations, all zeros being disregarded.

The combinations are:

0 001	U1
0 010	U2
0 011	U3
0 100	U4
0 101	U5
0 110	U6
0 111	U7
1 000	V
1 001	V1
1 010	V2
1 011	V3
1 100	V4
1 101	V5
1 110	V6
1 111	V7

The execution of the combinations are as follows:

5.5.1 The Combinations U1–U7.

The execution of the U1–U7 combinations depends on the state of the manual keys 1–7. (See 15.1.9.) Note that when V = 0 it is called U, i.e. U = $\overline{V}$.

5.5.2 **V1—Test A Negative.**

V1 tests on the sign digit of A. The test succeeds if the sign digit is 1. If the sign digit is 0, the test fails and the instruction is regarded in the machine as an A instruction. (See 5.6.)

5.5.3 **V2—Test B Negative.**

V2 tests on the sign of B and succeeds if it is 1. If a 0, then the test fails and the instruction is again regarded as an A instruction. (See 5.6.)

5.5.4 **V3—Test A Zero.**

V3 tests to see whether there is a 1 anywhere in the A accumulator. If a 1 is present the test succeeds: if all the 33 digits are zero, then the test fails. In this test the extra digit of A is disregarded.

5.5.5 **V4—Test Least Significant Digit of B.**

V4 tests on the least significant digit of B and succeeds if it is a 1. If it is a 0, the test fails and the instruction is regarded as an A instruction. (See 5.6.)

5.5.6 **V5, V6, V7.**

These combinations are not test digits and are used for the control of external equipment.

5.5.7 **The V Digit.**

The combination V has a special use. It controls the release of the 'carry' from the head of B to the tail of the A accumulator. Arithmetic in B may produce a 'carry'. This is trapped in a 'carry trap' and is only released into A when there is a subsequent instruction containing the V digit. (For amplification see 7.3.3.)

5.6 **The Opening Symbols: A and X.**

When writing instructions, only those digits having the value of 1 are written, the others being omitted. In writing the two addresses (i.e. drum and register) there is a convention that the drum address is written first. Thus the instruction A100BC5 means: add the contents of drum location 100 and the contents of register 5 into the cleared B accumulator. This can be abbreviated to: (100) + (5) ⟶ cleared B.

Instructions written in this way are interpreted by a Normal Input Program, which is normally kept inside the store. For example, when B is read, a 1 is automatically placed in the correct position in the instruction which is being formed in the machine. For this reason, the order and placing of these letters in an instruction word is not important. It is simply convenient to place them between the two addresses in order to separate the addresses. Addresses can be separated by a point. Thus A100BC5 = A100CB5 = A100B5C = ABC100.5. But the opening symbol A must go at the beginning since it marks the beginning of the instruction; moreover, the end of one instruction is marked by the beginning of the next.

In the case when A = 0, there is a special symbol called X. There are, therefore, two types of instructions: A and X.

Example: X100BC5 = next instruction in 100; (5) ⟶ cleared B.

To avoid ambiguity between drum and register addresses, drum addresses of less than 32 are signified by three digits, e.g. 031. Thus X004C4 = next instruction in drum location 004; (4) ⟶ cleared A.

5.7 **The W Digit.**

This is not normally provided directly by the programmer, but is supplied internally by the automatic action of the Normal Input Program.

If an instruction is associated with a word located in the main store, its execution must be delayed until the drum has rotated to its correct position. Under these conditions the W digit is absent. When the W digit is present the reading and writing circuits associated with the main store are inhibited or blocked, and the instruction is executed immediately.

This is used in *repeat instructions*, in which the drum address, made redundant by the inclusion of the W digit, can be used as a counting device. (See 6.8.)

6.1 The 'C' and 'E' Control Registers.

Words can come into the control unit from drum locations and registers. They are first added together before entering the 'C' control register, which is a shifting register. When the complete word is in 'C', the position part of the drum address is examined and the word is held circulating in 'C', until the drum is orientated into position. It is then allowed to jump sideways into the static register 'E', from which switching occurs during the 50 microseconds interval between words, and is held for the whole of the successive word-time.

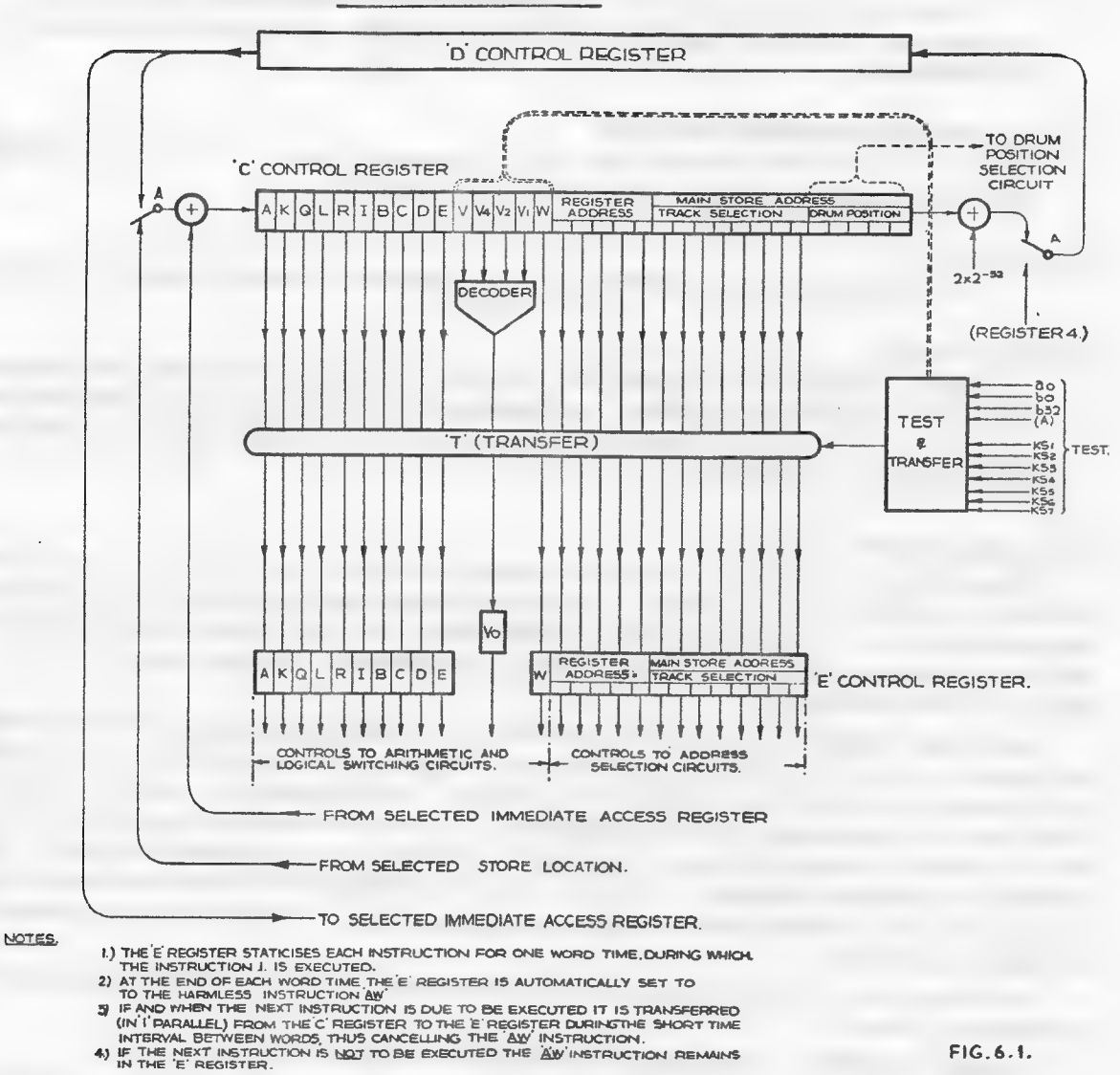


FIG. 6.1.

6.2 The Test and Transfer Box.

The test-and-transfer box, which allows this transfer, also examines the decoded combinations U1–U7, V1–V4, and allows or does not allow the transfer according to their state and to the corresponding state of the switches or arithmetic unit.

6.3 W = 1.

The test-and-transfer box also examines the W digit. (See also 6.5.) If W = 1, the word entering 'C' is immediately allowed to jump into 'E' and is executed. The output from the drum is now blocked and can neither be 'read from' nor 'written into'. Thus an instruction with W = 1 has a redundant drum address, but the special use as a count can, in this case, be made of it. (See 5.7 and 6.8.) It is worth noting again that the W digit is not usually entered into an instruction by the programmer but automatically by the Input Program in those instructions not requiring waiting for the drum. Note that W is examined in 'C' by the test-and-transfer box and is also staticised in 'E' to keep the drum inhibited during the word-time in which the instruction is executed.

6.4 The Action of X and A Instructions: The 'D' Control Register.

If the instruction entering 'C' is an X instruction, it passes on serially into 'D', as well as jumping into 'E'. In the process 2 is added to the drum address.

If, however, the word entering 'C' is an A instruction, then the next word from the drum will have as its destination the Arithmetic Unit. In this case, the next word for 'C' comes back from 'D' by the action of the A switch at the entrance to 'C'.

The reason 2 is added to the drum address when in 'D' is that when it returns to 'C' it can add in the next address which is the most optimum available without detriment to timing. To this extent Zebra can be said to have an optimising facility built into it.

X and A instructions can be best clarified in the following examples :

(1)

Drum location	Program instruction
99	X100
100	ABC101
101	Constant

B Accumulator	'C'	'D'
Constant	X100 ABC101	X102

(2)

Drum location	Program instruction
99	X100BE4
100	ABC101
101	Constant

B	C	D
Constant	X100BE4 ABC101 X102BE4	X102BE4 X104BE4

Notice that the instruction returns again from D and may bring with it a second action of some of the operational digits. Thus, in the second example, (B) $\longrightarrow$ 4 again after the execution of the A instruction. For this reason X100BE4 is an example of a *pre-instruction.* (See 6.10 and 7.4.)

Below is illustrated the 'pattern' of operations :

C	D
$X_1 \ldots$ $X_2 \ldots$ A.... $X_2 \ldots + 2$	$X_1 \ldots + 2$ $X_2 \ldots + 2$

When an X instruction passes from 'C' to 'D' and has 2 added, it remains in 'D' until either (1) it returns to 'C' after a subsequent A instruction, or (2) is wiped out by another X instruction from 'C' immediately following it.

6.5 'Failing' Test Instruction: AW.

Let us now consider an example of a test instruction which fails :

If fails

99	X100
100	X101V2
101	.....
102	

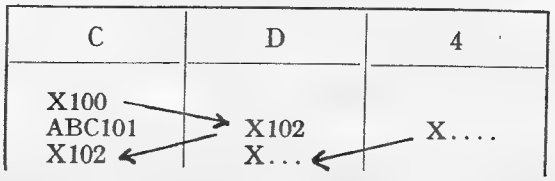
If succeeds

B	C	D
	X100 X101V2 X102	X102

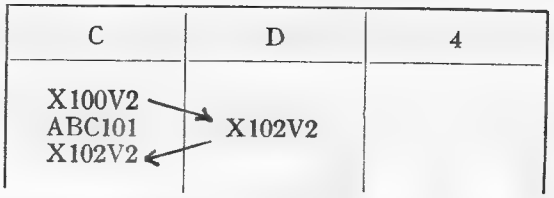
If the test succeeds the instruction would jump to 101. When the test fails, however, as above, the instruction X101V2, which tests on the sign of the B accumulator, does not go into 'E'. Instead the test-and-transfer box automatically provides the instruction AW. This is harmless, but because it is an A instruction X102 returns to 'C' from 'D', i.e. when the test fails there is a jump to the next-but-one location.

6.6 Connection between 'D' and Register 4.

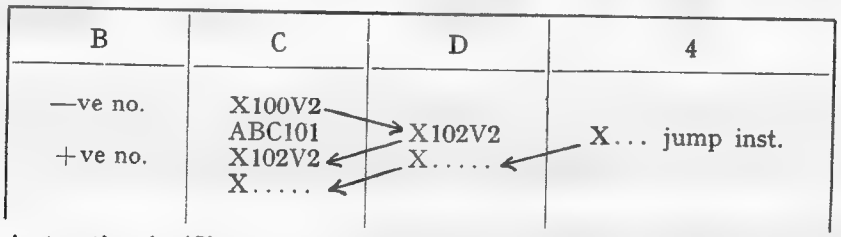
When the contents of 'D' pass into 'C' after an A instruction, it also happens that the contents of register 4 pass into 'D'. Abbreviated, this can be written as : when (D) ⟶C, then (4) ⟶D.



Let us take as a further example the case of the failing test instruction which follows an A instruction :



In 6.5 we have seen that a test instruction which fails is interpreted as an A instruction. So here we have two A instructions following each other, and there is the danger that there is no X instruction available in 'D' which can return to 'C'. But (4) can pass into 'C'. So if, say, an appropriate jump instruction had been previously written into 4, then this would return to 'C' via 'D'. In this example it would immediately follow the AW instruction which has automatically replaced the failed test instruction.



The jump instruction in 'C' can now be executed, causing the machine to jump to a further instruction in any specified location.

6.7 Calling in a subroutine.

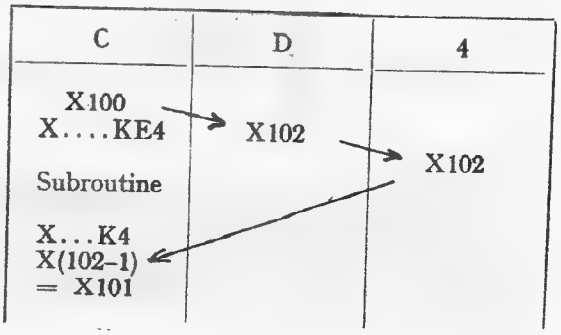
There are some processes in computation which occur very frequently, such as square-rooting, evaluating logarithms, and so on. To save the time and space of having to insert instructions to perform these operations every time they occur, they are written once and *called in* every time they are required. Such a program is called a subroutine. Subroutines are prepared on tapes and kept in a library.

To call in a subroutine it is necessary to jump to the appropriate location where the subroutine begins and to return to the next instruction in the main program after the subroutine has been executed. This is done by attaching KE4 (i.e. 4 or any other register) to the instruction which jumps to the subroutine and to end the subroutine with :

X next address K4
—1

These last instructions link the subroutine to the next instruction of the main program. The action of these instructions is described below :

99	X100
100	X address of subroutine KE4
101	.....



The instruction in 4, by means of which we have come back to the main program, is called in Zebra the *return instruction.*

6.8 Repeated Instructions.

It is required to repeat an A instruction which is in a register, say 5, a number of times, say 3. The instruction used is : X5K3, which is called the *repeat instruction.*

Now in this instruction the drum address and register address are interchanged. The W digit is automatically inserted by the Normal Input Program (see 6.3) and the drum address blocked. The drum address 3 is regarded now as a count of 3.

The instruction X5K3 is interpreted in the machine as an instruction XKW which has a register address of 5, and a drum address of 8192–2 × 3.

The action is as follows :

	C	D
	X5K8192–6	→ X5K8192–6 + 2
1	A100	= X5K8192–4
	X5K8192–4	←
2	A100	→ X5K8192–4 + 2
		= X5K8192 — 2
	X5K8192–2	←
3	A100	→ X5K8192 — 2 + 2
		= X5K8192 = X6K
	X6K	←

Notice here that X5K8192 = X6K. This is because $8192 = 2^{14}$ and there are only 13 digits located in the main store address ; so there is an overflow into the register address position and the register address is augmented by 1. (See Fig. 5.1.) Therefore X5K8192 is identically equal to X6K000.

The A instruction, which is the contents of register 5, has been repeated three times. The next instruction would, in this example, have been previously placed in register 6. There is a convention that jump instructions are underlined in a written program for the programmer's convenience, i.e. X5K3.

6.9 Some Simple Program Examples.

Here are some very simple examples of Zebra normal code programs. X instructions are basically 'take' instructions, and A instructions are similarly 'do' instructions. This take-do rhythm is typical of many computers, but it is possible to break the rhythm.

(1) Example : (300) + (500) + (700) ⟶ A.

100	X101	Take
101	A300C	Do
102	X103	Take
103	A500	Do
104	X105	Take
105	A700	Do
106	X107	Take
107	X200	Take

It is possible and clearly desirable to 'take' and 'do' simultaneously.

(2) Example : (5) + (7) + (12) + (14) ⟶ B.

100	X101BC5	(5) ⟶ cleared B
101	X102B7	(5) + (7) ⟶ B
102	X103B12	(5) + (7) + (12) ⟶ B
103	X104B14	(5) + (7) + (12) + (14) ⟶ B

This is a powerful feature of Zebra.

(3) This example shows the use of the I digit and the point :
(5) + (6) — (7) ⟶ A.

100	X101C5	(5) ⟶ cleared A
101	X102.6	(5) + (6) ⟶ A
102	X103I7	(5) + (6) — (7) ⟶ A

6.9.1 N and NKK.

X next address can be abbreviated and written as N. This is recognised inside the machine by the Normal Input Program. Thus the last example could be written as :

```
100 | NC5
101 | N6
102 | NI7
```

Similarly, A next address has an abbreviated form : NKK.

```
Example : 101 | ABC102      ≡   NKKBC
          102 | X002.1
          103 | A104CE15    ≡   NKKCE15
```

6.9.2 The L and R Digits.

The L and R digits shift both accumulators, which can also be coupled together for double length working. There is a special convention about clearing and shifting : shifting digits never appear in a simultaneously cleared accumulator. (See 5.4.6.)

```
Examples : A200CE5R : (A) ⟶ 5 ; (200) ⟶ cleared A ;
                      (B) is right shifted.
           NRBC3 :    (3) ⟶ cleared B ; (A) is right shifted.
           NRC2 :     (2) ⟶ cleared A ; (B) is right shifted.
```

The above examples show the action of R on single length accumulators. Notice that it is the unspecified accumulator which is right shifted here. The action of the L digit is similar.

Now the A accumulator has an extra digit. (See 2.4.) This is required, as has been seen in 2.5, for the multiplication process. The digits of A are labelled $a_{-1}\ a_0\ a_1 \ldots\ldots\ a_{32}$, where a_{-1} is the extra digit. On right shifting, the extra digit moves into the sign digit position, and the extra digit position is refilled by a copy of its previous contents, i.e. $a_{-1} \longrightarrow a_0$; (a_{-1}) new $\longrightarrow$ (a_{-1}) old.

6.9.3 The Combination LR.

The combination LR, which is logically meaningless, has, however, a special meaning. It is decoded to give a facility used in the multiplication process. (See 7.4.)

6.9.4 Double Transport.

Notice that the D and E digits used separately enable instructions to be given which allow simultaneous reading from and writing into the arithmetic unit.

```
Example : (1) AD300BC6 : D writes (B) ⟶ 300
                         B is cleared
                         (6) ⟶ B
          (2) A300BCE6 : E writes (B) ⟶ 6
                         B is cleared
                         (300) ⟶ B
```

6.9.5 A Note on the I Digit.

The I digit has no influence on the control unit or on writing into the store.

```
Example : (1) A300IBCE6 : writes (B) ⟶ 6
                          I only affects (300) ⟶ B
          (2) A300IK5 :   I only affects A300
                          i.e. makes (A) = —(300)
```

6.10 Pre-instructions.

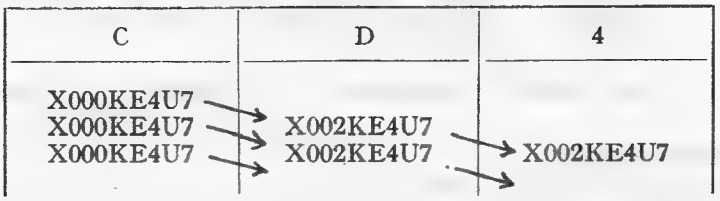
Pre-instructions are X instructions, whose effect is not of use until they return to 'C' from 'D' in later sequence.

An example can be seen in the multiplication program 7.4.

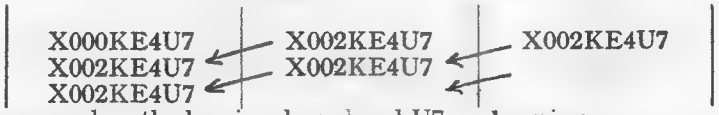
7.1 The Dynamic Stop.

When Zebra is switched on and waiting to be started, it executes over and over again a single instruction. This instruction is permanently recorded in drum location 000. The instruction is: X000KE4U7 ; it is used in conjunction with the manual key U7, which is the start key.

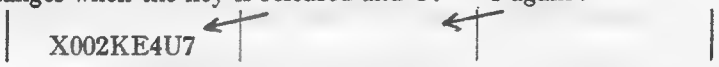
The action of this instruction, called the dynamic stop, is an example showing the combined action of the C and D registers and register 4. The action is as follows:



The above sequence takes place when the start key is not pressed, i.e. when U7 = 1, and continues until the key is depressed. When this is done, and U7 = 0, the action becomes:



This sequence changes when the key is released and U7 = 1 again:



When the last instruction is executed it causes a jump to location 002, which contains a further instruction. This mechanism returns when the 'clear' key (see 15.1.6) is pressed during some action of the machine. The clear key causes zero, i.e. the instruction X000, to come into the 'C' register and hence a return to the dynamic stop.

7.2 Repeated Instructions: Multiple Shifting.

It has been seen in 6.8 that an instruction of the form X5Kp causes (5) to be repeated p times. This device can be fitted into a program as follows:

100	NKE6
101	X5Kp

Here the A instruction to be repeated is in register 5 and the return instruction X102, placed by the action of KE6, is in register 6. An example is *multiple shifting*:

100	NKE6	5	AR
101	X5Kp(R)	6	X102

If the R digit is included in the X5Kp instruction, then 2p + 1 shifts occur; with no R, p shifts occur.

A complete piece of program, which sets AR in 5 as well, is shown below:

100	
101	NE5
102	A103C
'103	AR
104	NKE6R
105	X5KpR

A	C	D	Remarks
	X102E5		
	A103C	X104E5	
AR	X104E5		(A) = AR ⟶ 5
	X105KE6R	X106E5	X106E5 ⟶ 6
	X5KpR		
	AR	X5Kp—1R	
	X5Kp—1R		
	AR	X5Kp—2R	
	until X6KR		(6) = X106E5 → C

In this example the return instruction is X106E5.

The *comma notation* on location 103 indicates that the contents of the location are to be regarded as a constant, i.e. they are not executed directly from location 103.

7.3 Double Length addition and Subtraction: Use of Carry Trap.

7.3.1 Double Length Addition.

Example:

There is a double length number in registers 7 and 8 (head and tail, respectively) and a double length number in A and B. The double length sum is required in 7 and 8.

Program:

	Remarks.
NB8	(8) ⟶ B and added to (B).
N7V	(7) ⟶ A and added to (A). The V
NBE8	digit controls the carry from B to A.
NE7	(B) ⟶ 8; (A) ⟶ 7.

Rule: The carry produced by addition in B is transmitted to A by V. (See 5.5.7.)

7.3.2 Double Length Subtraction.

Let us suppose that the situation is the same as above, but the double length difference is required.

Program:

	Remarks.
NIB8	(8) ⟶ B and is subtracted from B.
NI7V	(7) ⟶ A and is subtracted from A. IV
NBE8	controls the borrow from B to A.
NE7	(B) ⟶ 8; (A) ⟶ 7.

Rule: The borrow produced by the subtraction in B is transmitted to A by IV.

Hence V or IV follow $\bar{I}$ or I arithmetic in B, respectively.

7.3.3 The Carry Trap: Normal Action.

The normal action of the carry trap can be shown as follows:

Arithmetic in B	Release in A
$\bar{I}$: Carry 0	V: Carry 0
$\bar{I}$: Carry 1	V: Carry 1
I: Borrow 0	IV: Borrow 0
I: Borrow 1	IV: Borrow 1

The entrance to and exit from the carry trap are influenced by the I digit, so that if we show the contents of the trap as well, the diagram becomes more fully:

Arithmetic in B	Contents of Trap	Release in A
$\bar{I}$: Carry 0	0	V: Carry 0
$\bar{I}$: Carry 1	1	V: Carry 1
I: Borrow 0	1	IV: Borrow 0
I: Borrow 1	0	IV: Borrow 1

7.3.4 The Carry Trap: Exceptional Cases.

The table below gives the exceptional cases which break the rule:

Arithmetic in B	Contents of Trap	Release to A	
$\bar{I}$: Carry 0	0	IV: Borrow 1	X
$\bar{I}$: Carry 1	1	IV: Borrow 0	
I: Borrow 0	1	V: Carry 1	X
I: Borrow 1	0	V: Carry 0	

It can be seen from this that borrow 0 is inverted to carry 1, etc. Of particular importance are those entries marked X. An instruction which adds 0 to B is harmless, but a subsequent IV instruction can be used to subtract ε from A.

An instruction subtracting 0 from B will produce borrow 0 and is harmless, but a subsequent V instruction can be used to add ε into A. An example of this is used in the multiplication program. (See 7.4.)

7.4 Double Length Multiplication: Use of Register 15.

The method of multiplication used is precisely that described in 2.5. The fundamental operation is that of right shifting the accumulators and adding the multiplicand into A if the least significant digit of B before the shift was 1. This facility is provided by the LR combination. LR causes the product of the least significant digit of B and register 15 to go into the A accumulator, i.e. LR causes $b_{32} \times (15) \longrightarrow A$. Register 15 can therefore be used to contain the multiplicand. Register 15 is also used in connection with the XD facility in the division process. (See 8.1.)

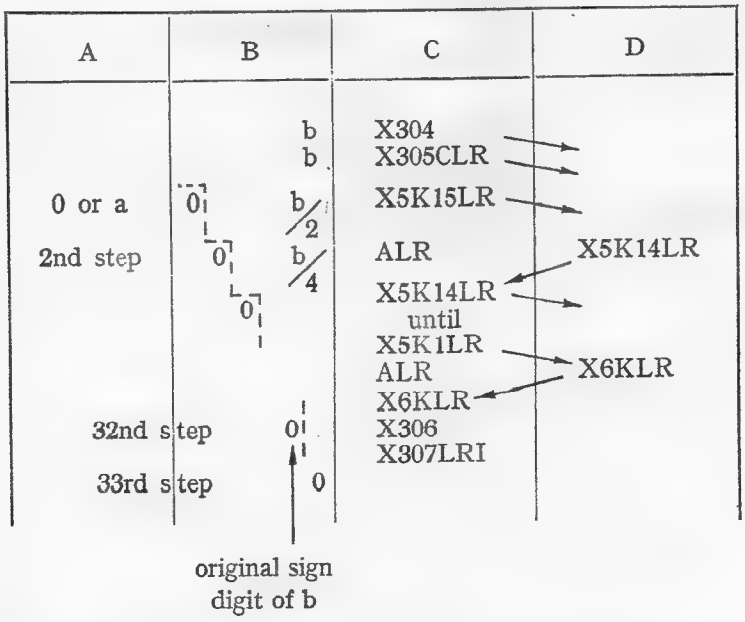
7.4.1 The Action of the Multiplication Program.

The basis of the program is :

303	N	5	ALR
304	NCLR	6	X306
305	X5K15LR	15	multiplicand = a
306	NLRI	B	multiplier = b

Notice that LR occurs in both the repeat and the repeated instructions. In this manner we can obtain the sign digit and 64 digits of the product ; the 66th digit, which is the least significant digit of B, is zero.

The detailed action is :



The final step is done negatively. This takes care of the sign digit. If b is positive, nothing is subtracted from the double length product at the 33rd step. If b is negative, the multiplicand is subtracted from (A) at this stage.

7.4.2 The Complete Multiplication Program.

The complete program is :

300	NIB
301	NKKCE15
"302	ALR
303	NE5V
304	NLRCKE6
305	X5K15LR
306	NLRIBK4
"307	—1

This could be used in another program by having for instance,

199	X200
200	X300KE4B,

and this would give a rounded answer, correct to 32 binary digits, in A.

If 200 | X301KE4B
this would give a double length answer in AB.

We assume that the multiplier b has already been placed in B, and the multiplicand a in A.

7.4.3 Unrounded Product.

The detailed action for a double length answer is :

A	B	C	D	4	6	Remarks
a	b	X200				
		X301KE4B	X202			
		A302CE15	X303KE4B			carry trap set to 0
ALR		X303KE4B	X202			a ⟶ 15
		X304E5V				
		X305LRCKE6	X306E5V			ALR ⟶ 15. V has no effect, since 0 in carry trap
0 or a	0!	X5K15LR				
2nd	step	ALR	X5K14LR			
3rd	step	X5K14LR				
		until				Double length product built up in A and B in 33 steps
		ALR				
		X6KLR				
		X306E5V				
		X307K4LRIB				
33rd	step	X201				
		(= X202 — 1)				

7.4.4 Rounded Product.

The action for a single length rounded product is :

A	B	C	D	4	6	Remarks
a	b	X200				
		X300KE4B	X202			
		X301IB				
		A302CE15	X303IB			carry trap set to 1
ALR		X303IB				
		X304E5V				
ALR + ε		X305LRCKE6				
0 or a	1¡					
			until			
		ALR				
		X6KLR				
		X306E5V				
xxx . . . xxx	xxxx . . . x	X307K4LRIB				
xxx . . . xxx	xxx xx					ε is released into A by V

After the ε is added to A by the V digit it is immediately shifted into B, so the effect is the same as adding a 1 into the head of B at the 33rd step. This gives the correct round-off to the product in A.

7.5 Relative Drum Locations.

In the program examples previously given we have numbered drum locations with 100, 101, etc. This is done purely for purposes of illustration and convenience. In fact, when a program is actually written, drum locations are numbered in a relative form with P, P1, P2, etc. This is interpreted automatically by the Normal Input Program which chooses a drum location relative to P which is the most convenient. (For amplification see 11.4.)

8.1 The XD Facility.

The combination XD is used in the division process. An instruction containing this combination causes, in addition to the separate actions of these digits, the contents of register 15 to go into the specified accumulator, i.e. XD causes (15) ⟶ specified accumulator.

Example : X100K5D : (A) ⟶ 100.
(A) + (15) ⟶ A.
(5) ⟶ C = 'C' register.

D also causes writing into the drum even though there is an X instruction. With the XD combination work can still be done in A even though the contents of the specified register are going into C. It is usual to have (register) ⟶ C in an XD instruction, otherwise nothing can reach C and the machine stops. The instruction XD100, for example, will cause a stop.

8.2 Division : Short Process.

The central process in the division of fraction by fraction is as follows : Let us assume that the dividend a is in A, and that minus the divisor b is in register 15, i.e. a = (A) ; —b = (15). We wish to find $\frac{a}{b}$:

100	X4KDQp	4	AI15QV1
101		5	X101
		15	—b

In register 4 we have the repeated instruction AI15QV1, which is also a test instruction. Test instructions are signified by the dotted underline notation. V1 tests A negative (see 5.5.2). Register 5 contains the return instruction X101.

A	B	C	D
a		X4KDQp	
a — b	ε	AI15QV1	X4KDQp—1
		X4KDQp—1	
a — 2b	2ε	AI15QV1	
a — b	ε		
.....	.....	.....	...and so on
remainder	quotient		

Tests if (A) < 0. If not, the test fails.
When (A) < 0 then b has been subtracted once too much. This is compensated by I15Q.

Such a short division process is used in binary to decimal conversion.

The mechanism is : (1) Subtract.
(2) Test if positive or negative.
(3) If positive, then write 1.
(4) If negative, then recover.

8.3 The Normal Division Program.

This introduces register 23 which permanently contains the sign digit ; i.e. (23) = 2^0 as a constant.

In the normal division program, the process described above works more quickly since the remainder after each step is left shifted before the next subtraction. (See 2.4.) The numerator must be smaller than the denominator. The rounded quotient in A and B is required.

The full program is :

100	NIBC3
101	X103IC2V2
102	
103	NBE15
104	NE6
105	A106CE7R
'106	AQI15V1
107	A108CE5LQ
'108	X114I15
109	NIC7V
110	X112QI15V1
111	
112	X5K32LDQ
113	
114	NRB23Q
115	NC3
116	NK4
117	—1

Uses registers :

4	Return instruction
5	AQI15V1
6	X114I15
7	dividend a
15	—divisor b

The program is called in by X100KE4.

A	B	C	D	Analysis
a	b	X100KE4	return instr.+2	r.i. + 2 ⟶ 4
		X101IBC3 →	X103IBC3	Clears B ; adds —(3) ⟶ B. (Register 3 = (B).)
	—b	X103IC2V2 →		Clears A ; adds —(2) ⟶ A. (Register 2 = (A).)
—a				We show here the case where b > 0. With b < 0 we should not have changed a, and b would be restored.
		X104BE15 →		—b ⟶ 15
		X105E6		Pre-instruction.
		A106CE7R →	X107E6	(A) = —a → 7 (+a if b < 0).
AQI15V1	—b/2			Constant ⟶ cleared A.
				Right shift b, i.e. divide by 2.
		X107E6 →		
		A108CE5LQ →	X109E6	AQI15V1 ⟶ 5 ; A cleared ;
X114I15	—b' + ε			constant ⟶ A.
				B is left-shifted again so that —b is restored, but with least significant digit now 1.
		X109E6		X114I15 ⟶ 6. Carry trap has been set to no carry by the Q digit.
		X110IC7V →		—(7) —ε ⟶ cleared A.
a — ε			X112IC7V	I.e. a —ε ⟶ A (—a —ε if b < 0). Because of IV, the 'carry 0' is interpreted as 'borrow 1' (see 7.3.4).
		X112QI15V1		At this step (A) + (B) can be regarded as a — ½ b'ε in A.
a — ε	—b' + ε			

The final ε in B can be regarded as the first digit in the quotient assuming that we are dividing (a — ½ b'ε + b) by b. It is as if we had already done the first subtraction of b and we must look to see if the result is +ve or —ve, to leave alone or add b in again, respectively. In the latter case we must also remove the "one" from the quotient. This is the result of the last instruction appearing in C. If it is not executed, then X112IC7V returns harmlessly from D.

A	B	C	D	Analysis
$2(a - \frac{1}{2}b'\varepsilon + b) + (15)$ | XX	Continues until	X5K32LDQ AQI15V1	X5K31LDQ	(15) are added to the left-shifted accumulator, and now we have two digits of the quotient (marked XX). At each stage, if the subtraction of b "won't go", it is removed and the quotient digit put zero as above.
		AQI15V1 X6KLDQ	X6KLDQ	At this stage we have done one step too many and have in B : $2\frac{(a - \frac{1}{2}b'\varepsilon + b)}{b} + (15) =$ $2(1 + q - \frac{1}{2}\varepsilon) + (15)$ where q represents the quotient. We therefore remove (15), right shift b, and add $1 + \varepsilon$.
	$2(1+q-\frac{1}{2}\varepsilon)$ $q+\frac{1}{2}\varepsilon$	X114I15 X115RB23Q		This is done with the instruction X115RB23Q. This now gives $1+q-\frac{1}{2}\varepsilon+1+\varepsilon = q+\frac{1}{2}\varepsilon$, which is the correct rounded quotient.
$q + \frac{1}{2}\varepsilon$	$q + \frac{1}{2}\varepsilon$	X116C3 X117K4 Return instruction		This is also placed in A. X117K4 brings the return instruction from 4 into C. Hence a return to the main program.

The quotient is found as $2 + q + \frac{1}{2}\varepsilon$. The 2 is lost off the front in the case of a positive quotient, but serves to give a correct result in the complementary notation if the quotient is negative since $-q$ is represented as $2-|q|$.

The time of execution for the division subroutine is 35.7 ms.

8.4 Block Transport.

It is very important to be able to bring the contents of the drum locations into registers.

I.e. $n \longrightarrow m$
$n + 2 \longrightarrow m + 1$
$n + 4 \longrightarrow m + 2$, etc., where n represents the drum location and m the registers. These use the instructions An CEm -1
An $+$ 2CEm,
An $+$ 4CEm $+$ 1, respectively.

This process may be performed by using *a repeated instruction which must be altered each time.* This is achieved by placing the instruction to be altered in an accumulator and by altering it there.

Example : Put (n), (n + 2), etc., into registers 6–15, inclusive.

The program is :

100	NC5		
101	ABC102	=	NKKBC
"102	X002.1		
103	AB104CE15	=	NKKCEB15
"104	ACnE5		
105	NKE4		
106	X3K11BD		

This program makes use of the XD facility mentioned above (8.1). The repeated instruction to be altered is ACnE5. Note that register 3 = (B) and register 1 contains 1. The previous contents of 5, if any, are preserved.

Detailed action :

A	B	C	D	Remarks.
		X101C5 → ABC102	→ X103C5	X002.1 ⟶ cleared B.
	X002.1	X103C5 → AB104CE15	→ X105C5	X002.1 ⟶ 15.
	ACnE5	X105C5 → X106KE4	→ X107C5	ACnE5 ⟶ cleared B.
		X3K11BD		X107C5 = return instruction ⟶ 4.
				(B) = ACnE5 ⟶ C.
				XD sends (15) = X002.1 ⟶ B.
				This is added to ACnE5 in B to become ACn + 2E6.
	ACn + 2E6	↘ ACnE5	↘ X3K10BD	(n) ⟶ cleared A.
		X3K10BD		(B) = ACn + 2E6 ⟶ C.
(n)				XD adds (15) = X002.1 ⟶ B.
		↘	↘	Thus we have ACn + 4E7 in B.
	ACn + 4E7	ACn + 2E6	X3K9BD	(n) ⟶ 6 ; (n + 2) ⟶ cleared A.
(n + 2)		X3K9BD ←		
		etc.		and (n + 2) ⟶ 7
		until		(n + 4) ⟶ 8, etc.
		X4KBD		return instruction from 4 ⟶ C.
		X107C5		

8.5 **Repeating Blocks of Instruction.**

Let us suppose there is a block of instructions in locations 200–220 which has to be repeated p times.

The program is :

96	N
97	NKKC
'98	X200KE6
99	NE4
100	NKE5
101	X4Kp

The block of instructions which are to be repeated must end with X6K in this instance.

Detailed action :

A	B	C	D	Remarks
		X97 → A98C	→ X99	X200KE6 ⟶ cleared A.
X200KE6		X99 ←		
		X100E4 →		X200KE6 ⟶ 4.
		X101KE5	X102E4	X102E4 ⟶ 5.
		X4Kp →		
		X200KE6	X4Kp—1	X4Kp—1 ⟶ 6.
		block ending with X6K		
		X4Kp—1		X4Kp—1 = (6) ⟶ C.
		until X5K		
		X102E4		Return instruction from 5 and jump back to program.

8.6 Short Multiplication.

8.6.1 Multiplication by Small Fraction.

There is no need to call in the multiplication subroutine every time that multiplication is required. For instance, if we wish to multiply a number by a fraction which has only a few digits in the most significant end, the multiplication can be performed very simply in the following way.

Example : We wish to multiply the number a by 0.1011.

We assume that a = (5).

The program is :

```
NC5     1
NR5     1
NR      0
NR5     1
NR      0
```

The multiplication is done in an accumulator—in this case A—by right-shifting a and adding in (5) corresponding to the digits of the fraction. In this case the process takes 1.5 ms., which is five word times. Notice that capacity can be exceeded twice and use made of a_{-1}. (See 6.9.2 and 2.5.)

If it is required to multiply by a fraction which has a few digits in the least significant end, it is better to use the B accumulator.

Example : (5) = a. Required : a × 0.00 ..... 1011001.

The program is :

```
NBC5    1
NLBV    0
NLB5V   1
NLB5V   1
NLBV    0
NLBV    0
NLB5V   1
NV
```

Notice that there is a special action associated with LV. The carry digit is released *before* the shifting, i.e. it goes in the a_{31} position. It is this action which permits this double length working.

The LV action does, however, present difficulty with instructions of the form AnLmV. Without the L (i.e. An.mV) it is possible to release the carry digit into the carry part of the pre-adder of A during the first digit time (i.e. the least significant digit time) because there is then no carry from a previous step as a result of adding (n) and (m). But if (n) and (m) each contain a least significant one there is a carry produced to the next stage ; and if the carry trap contains one this also requires entry at the same time and the same point. The carry part of the pre-adder cannot accept both at once and one of them is lost.

Thus if (n) is odd, (m) is odd, 1 = (carry trap) } AnLmV will fail.

8.6.2 Multiplication by Small Integer.

Similarly there is no need to call in the subroutine if we wish to multiply by a small integer. This can also be done very simply.

Example : Multiply (B) × 10.

The program is :

```
NLBE5   2B ⟶ B
NLB5    5B ⟶ B
NLV     10B ⟶ B
```

8.7 Conjugation or Logical Product.

The conjugate (a, b) has ones where a and b both have one and zero elsewhere.

Example :

```
1011011 = a
1001101 = b
-------
1001001 = conjugate (a, b)
-------
```

The logical product can be used for separating a part of an instruction.

Example : We wish to separate the register address of an instruction word.

Instruction word	:	01.....	11.....0	10101	10.....1.....01
Mask	:	00.....	00.....0	11111	00.....0.....00
Conjugate (instruction, mask)	:	00.....	00.....0	10101	00.....0.....00

The conjugate of instruction and mask gives the word containing only the register address. We can imagine the 'mask' as being placed directly on top of the instruction word, masking what is not wanted and making transparent the part required—in this case the register address part.

The conjugate of (A) and (B) is always contained in the pseudo-register 24. Thus a program to find the register address part of the word in drum location n, say, would be as follows :

```
NKKC
X000.31  = mask
ABCn
X
NC24        conjugate ( (A), (B) ) ⟶ A
```

An extension to the logical product is required in looking for *equality*, i.e. to obtain a word having ones where both words are the same. This word C can be expressed as : C = conjugate (a, b)+conjugate (inverse a, inverse b). The *inverse* of a word is obtained by interchanging ones and zeros. The inverse of register m can be obtained with the single instruction NBICmQ.

Now since register 24 always contains the conjugate ((A), (B)), it is not possible to write into it. E24, which is therefore logically impossible, has, however, a special meaning : E24 in an instruction obtains the conjugate of the specified drum location and register 5. (Always register 5.) I.e. AnE24 sends conjugate ((n), (5)) ⟶ A. This means that with a single instruction word and with a suitable mask in register 5, we can extract part of a word from the store.

Example : We wish to look up a drum location which contains a specific part. We assume minus the required part in register 15, and the mask for the part in register 5. The instruction AmE24Q is in B.

The one instruction we use is : X3KpDQCV3. Notice again the action of the XD facility.

A	B	C	D	Remarks
—part reqd. m′ —part reqd.	AmE24Q Am + 1E24Q Am + 2E24Q	X3KpDQCV3 AmE24Q X3Kp—1QDCV3	X3Kp—1DQCV3	(3) = AmE24Q ⟶ C. XD brings (15) ⟶ A. m′ is the conjugated part of m.

When the part is found the test fails and (4) ⟶ C.

If there is no part found X4KDQCV3 finally comes into C, but in this case (A) ≠ 0. The address of the required part is the address in B minus 2. Note that we are running through the store at full speed. If we need to search through consecutive locations, we can use firstly even, then odd, locations.

In the example above we can see that the machine is automatically finding and obeying its own instructions from other parts of the machine, i.e. more instructions than ever appeared in the written program. We call this type of action "underwater" programming. Examples of this can be seen in the multiplication program (7.4), in repeating blocks of instructions (8.5), and elsewhere.

9.1 Input Instructions: Use of Registers 26–31.

The normal way in which instructions are input in Zebra is on five hole-per-character punched paper tape. There are six pseudo-registers connected with reading from the tape. These are registers 26, 27, 28, 29, 30, 31. They are most commonly used by the programmer in test programs (see Part IV), particularly in those which test the input reader, and are also included in the Normal Input Program. (See Appendix.)

CHART SHOWING EQUIVALENT NUMERICAL VALUES TO TAPE

TAPE 5 4 3 2 1	BINARY VALUE	DECIMAL VALUE
	0	0
1	1	1
10	10	2
11	11	3
100	100	4
101	101	5
110	110	6
111	111	7
1000	1000	8
1001	1001	9
1010	1010	10
1011	1011	11
1100	1100	12
1101	1101	13
1110	1110	14
1111	1111	15
10000	10000	16
10001	10001	17
10010	10010	18
10011	10011	19
10100	10100	20
10101	10101	21
10110	10110	22
10111	10111	23
11000	11000	24
11001	11001	25
11010	11010	26
11011	11011	27
11100	11100	28
11101	11101	29
11110	11110	30
11111	11111	31

FIGURE 9.1A

CHART SHOWING MURRAY TELEPRINTER CODE AND COMPUTER CODE WITH SYMBOLIC REFERENCE TO PUNCHED PAPER TAPE

TELEPRINTER CODE LETTERS	TELEPRINTER CODE FIGURES	TAPE 5 4 3 2 1	ZEBRA COMPUTER CODE
			0
E	3	1	1
LINE FEED		2	2
A	—	11	3
SPACE		4	4
S	'	5	5
I	8	6	6
U	7	7	7
CARR	RETURN	8	8
D	TAB	9	9
R	4	K	K
J	BELL	Q	Q
N	,	•	•
F		L	L
C	:	R	R
K	(	I	I
T	5	B, H	B, H
Z	+	C, S	C, S
L	)	D	D
W	2	E	E
H		T	T
Y	6	U	U
P	0	V	V
Q	1	N	N
O	9	A	A
B	?	X	X
G		+	+
FIGURES		—	—
M	.	Y	Y
X	/	Z	Z
V	=	P	P
LETTERS		#	#

FIGURE 9.1B

Associated with the input reader there is also a "staticising" register or "staticiser". The staticiser contains the contents of the previous character to the one actually standing under the reader. When a new tape is put under the reader, press the reader button. This clears the staticiser and gets rid of any harmful character which may have been left by a previous action. The tape is one step ahead of information being read.

Register 26 has the following action. It sends —ε to the appropriate accumulator if the 5th hole is represented in the staticiser ; it sends zero to the appropriate accumulator if there is no 5th hole represented in the staticiser. The 5th hole corresponds to the most significant digit of the character. —ε is sent because ones are supplied for the whole of the word time. It is possible to send +ε by using I26.

Thus, NIBC26 means : O ⟶ cleared B if 5th hole is absent ; +ε ⟶ cleared B if 5th hole is present. Similarly, registers 27, 28, 29, 30 signify the 4th, 3rd, 2nd, 1st holes represented in the staticiser, respectively.

Register 31 causes the input reader to step the tape on to the next character, and, in a parallel action, transfers information under the tape into the staticiser.

Example (1) :

```
NIC26
NLI27
NLI28
NLI29
NLI30
N31
```

This program reads the character from the tape into A and steps tape.

Example (2) :

```
NIBC26
NLIB27Q
NLIB28
NLIB29Q
NLIB30
N31
```

Character — 10 ⟶ B

This can, for example, now be tested for a digit.

Example (3) :

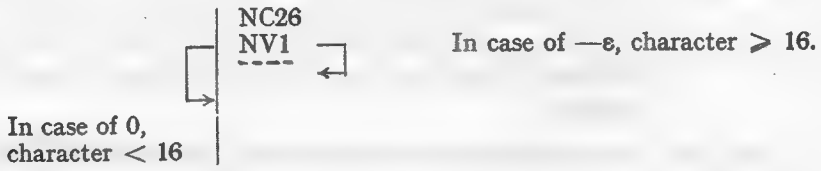
```
NIC26
NLI27
N11
NLI28
NLI29
N11
NLI30
N31
NE11
```

Character + 10 × (11) ⟶ 11

This is used in decimal to binary conversion.

Sometimes it is required to read the five holes in some other order, or to read only some of them.

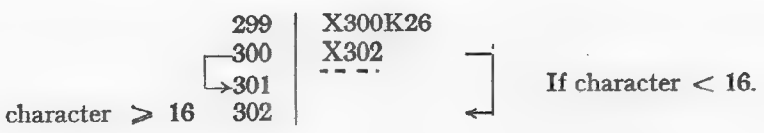
Example :



The 5th hole represented in the staticiser corresponds to the number 16 (see Fig. 9.1).

The hole can be directly read into the 'C' control register with a K digit. Since I cannot influence C, only —ε is given for the hole.

Example :



9.2 Output Instructions.

The mode of output in Zebra is usually output punch and/or teleprinter.

9.2.1 Punch : E26.

This is similar to the input method and uses registers 26–31 together with the E digit. The instruction E26 means : set the 5th hole for punching or not according to whether the sign digit of the A accumulator is 1 or 0. Similarly, E27, E28, E29, E30 are associated with the 4th, 3rd, 2nd, 1st hole, respectively. E31 means : punch and step the output tape.

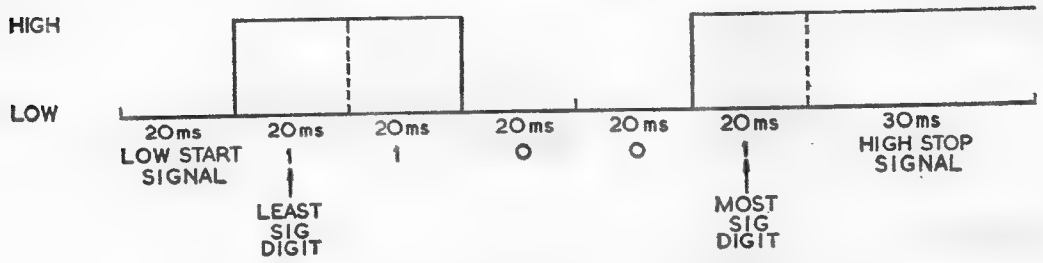
Example :

NLE26 NLE27 NLE28 NLE29 NLE30 NE31	Punches the character according to the five most significant digits of A which have been shifted out.

9.2.2 **Teleprinter : Register 25.**

The teleprinter is controlled by a seven-unit signal of the form :

FIGURE 9.2



The output of these time signals is completely controlled by program. This makes use of register 25, which causes a low or high signal to be sent to the printer according to whether the sign digit of A is 0 or 1. Also (25) = (A), so that selection of register 25 will double the contents of A. Thus, the instruction A25 sends a signal to the printer and doubles A.

Sometimes it is required to send a signal without doubling A. This can be done with the instruction E25.

Example : Send the character 10011 to the printer.

This requires 0. 11 11 00 00 11 11 in A. Register 5 contains the instruction A101.25.

Program :		
	100	AE25
	'101	Zero
	102	X5K13

AE25 : Sends low signal to the printer ; does not double A.

X5K13 : A101.25 = (5) is obeyed one revolution later. Sends low signal to the printer. So the printer is low for 20 ms. : this is the start signal. Then the high signal for 20 ms., and so on. Finally the signal is high for 20 ms. The high signal continues for another 10 ms. since a low signal cannot be entered for another revolution.

Since (5) is not obeyed again until the time of 101, the instruction X5K13 can be up to a track length further on. This "waiting time" is available for other instructions, such as setting a constant in A. This might be looked up in a list to correspond to a given letter or digit.

INPUT PROGRAMS

Chapter 10

There must be a small amount of information in the computer before it is capable of doing anything. The information is permanently recorded in track zero, i.e. drum locations 000–031. This information is known as the *Short Input Program.* It is linked to further stored information called the *Normal Input Program,* which interprets and carries out instructions written in the normal code.

10.1 The Pre-Input Program.

The Pre-input Program is entered into the machine manually from the control panel keys. It consists of two instructions, which are set in locations 000 and 8190. They are :

```
000  | X8190IB30
8190 | AD8191LK29
```

The Short Input Program itself must be put into the machine ; and this is entered off a specially punched tape with the aid of these two instructions.

10.2 The Short Input Program.

The complete Short Input Program (or S.I.P.) can be found in 22.4.1. In this paragraph we intend to illustrate its structure and the mechanism of its action.

It has already been seen in Chapter 7 that in location 000 there is an instruction of the form X000KE4U7, called the dynamic stop, which works in conjunction with the manual key U7. The key U6 is also used :

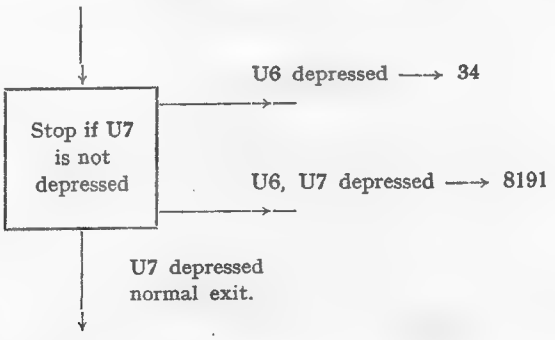


FIG. 10.1

Thus, besides the normal exit from the stop to the S.I.P., there are two other special exits caused by the use of U6.

(1) When the machine is on stop, and is restarted with the manual key U6, kept depressed, there is always a jump to *location 34.* This normally contains the *key address,* that is, the address of the beginning of the previously stored program. It will be seen later that this is a result of the way in which a complete program is stored. This special exit from the stop can be used to begin again at the beginning of a program, i.e. stop by clearing, and begin again by depressing the start key and releasing again with U6 depressed.

(2) If both U6 and U7 are depressed there is a jump to *location 8191.* An example of the use of this exit is given by the procedure adopted when the computer is left running unattended. (See 15.1.3.) In this case the machine is left with U6 and U7 depressed. A return to stop caused by a fault in a program will now run straight through the stop and jump to location 8191. Here begins a small program of six instructions which ignores tape until 100 consecutive blanks are read and then begins to read again. In this way the machine can leave one program and begin another.

10.2.1 **Location 32.**

Suppose that the first character read from the tape is > 1. This causes a jump from S.I.P. to location 32. When the Normal Input Program (or N.I.P.) is taken into the machine a jump instruction to the first track of the N.I.P. itself is written into location 32. This first track is called the "*Vertical Ladder*" (see 10.3 for amplification).

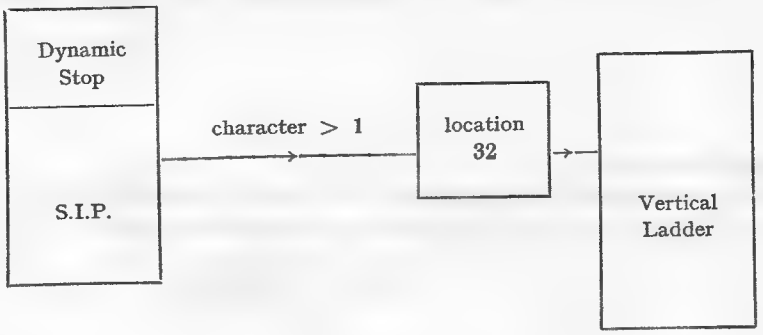


FIG. 10.2

10.2.2 **Character > 1.**

The S.I.P., therefore, does not accept tape which begins with a character > 1. It does accept and read tape beginning with a character 1. If 0, i.e. blank tape is read, the S.I.P. reads the next character, i.e. the S.I.P. is able to skip over blank tape before the first significant character is read. The diagram can now be enlarged to become :

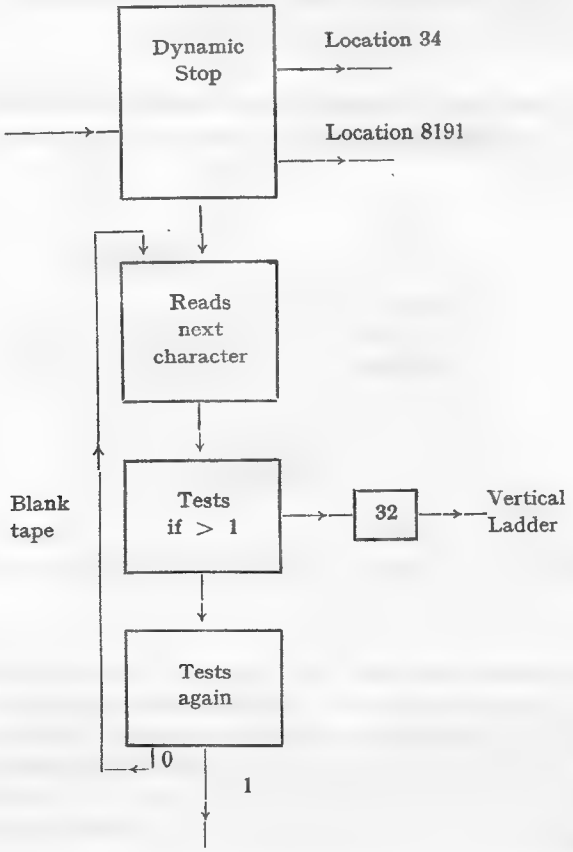


FIG. 10.3

10.2.3 **Binary Form.**

The Short Input Program can only read tape prepared in a special form, in which each word has to be written in its binary equivalent. Thus, for example, the instruction A003BC5V1 would be represented by :

100000110010010001010000000000011

These 33 digits, together with two extra holes on the tape to spare, can be written as seven five-hole characters. The extra holes are used for special purposes.

10.2.4 Parameter Indication Digit: Register 9.

On entering the S.I.P. proper, the seven characters are read and the 35 digits formed as: special digit + 33 binary digits + special digit. The last special digit is examined. If it is zero the binary word is unchanged, if it is 1 the binary word is modified by the contents of *register 9.*

This enables us to make use of a "parameter", that is, to make instructions with drum addresses relative to a fixed point (see 7.5), the point having previously been written into register 9. The special digit is called the *parameter indication digit* and register 9 is normally used as the *parameter register.*

10.2.5 Input Indication Digit: Register 11.

Now the first special digit is examined. If it is 1, the 33-digit binary word is written into register 11; if it is zero the binary word is stored according to the instruction in register 11. Thus the word written into register 11, by making this digit 1, is normally a "store instruction". This special digit is called the *"input indication digit".*

The input indication digit is therefore used to place a store instruction into register 11. In this way it is possible to begin a tape which has to be taken in by the S.I.P. with an instruction saying where the input must begin. Then storing from this point is sequential.

To end a short code tape, fill register 11 with a jump instruction instead of a store instruction. Then one further block of seven characters is read (normally this will be blank on the end of the tape) and instead of storing this extra word the jump is executed.

In this way the reading of the tape could end, and the machine would commence execution of the program or stop.

10.3 Vertical Ladder of the Normal Input Program.

The jump from location 32 (see 10.2.1) to the vertical ladder is modified according to the particular character > 1 which was read from the tape. Thus the vertical ladder is entered at a different point for each character. The scheme is as follows:

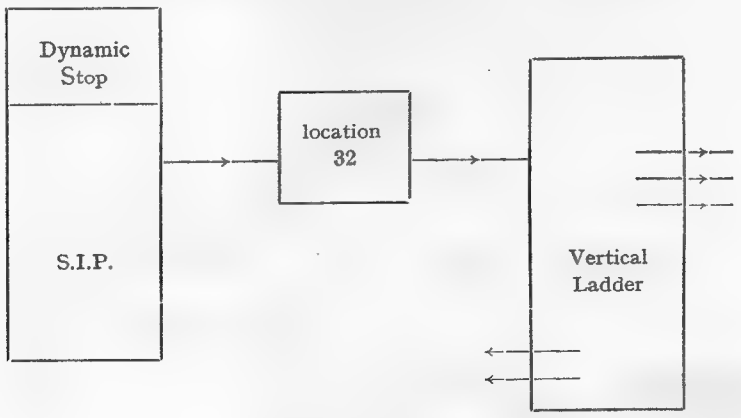


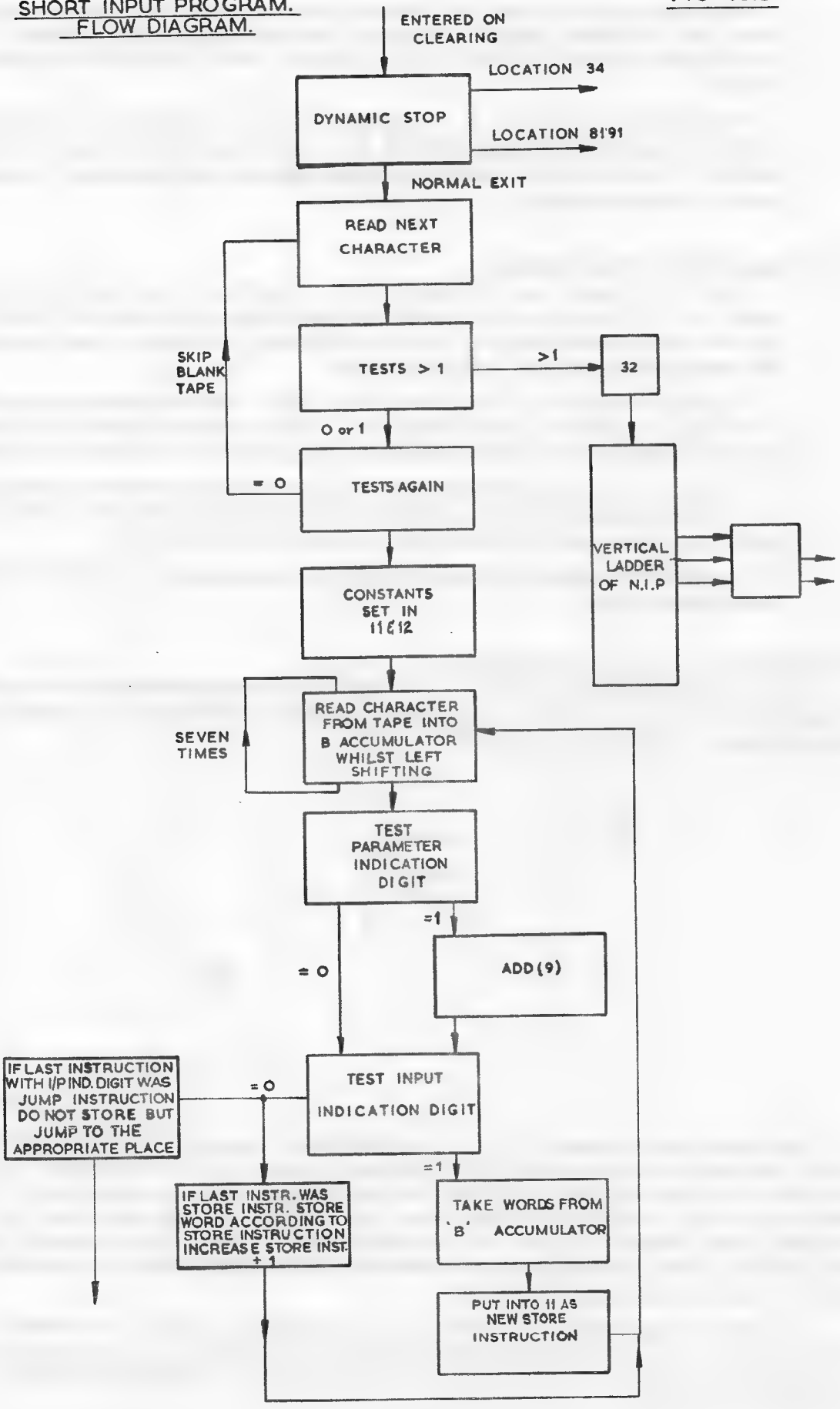
FIG. 10.4

The vertical ladder consists of a list of jump instructions jumping to appropriate parts of the N.I.P. for dealing with words beginning with the opening symbol which has been read. The first character on the tape is therefore the character for A, and this passes from the vertical ladder into that part of the N.I.P. for "constructing instructions". Similarly other opening digits have their exits from the vertical ladder (see Chapter 11). There are a number of positions in the ladder left empty (i.e. containing X000) for those characters representing letters or digits which do not normally start a word. These will cause the machine to stop.

A special meaning might be attached to one of these opening symbols: for example, a tape beginning with B is made to mean that what follows is in teleprinter code. The program which translates teleprinter code to the normal code will fill in the position in the ladder corresponding to B with a jump to this program.

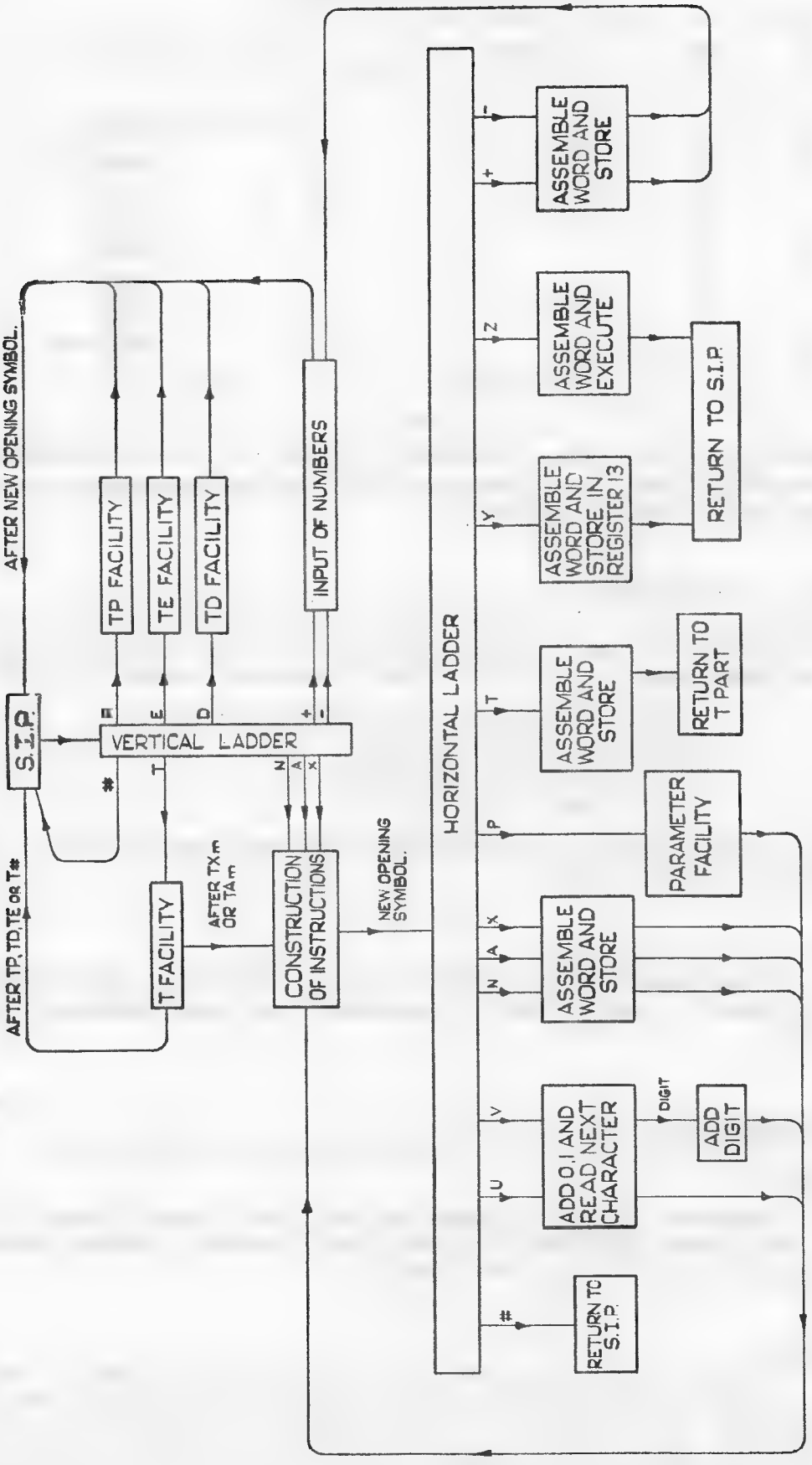
SHORT INPUT PROGRAM.
FLOW DIAGRAM.

FIG 10.5



THE COMPLETE N.I.P. BLOCK DIAGRAM IS AS FOLLOWS:-

FIG. 11.1.



CLASSIFICATION OF SYMBOLS FIG. 11.2

Digits	Supplementary Symbols	Opening and Closing Symbols	
0	K	T (see Chapter 12)	Opening
1	Q	U	Special opening
2	.	V	Special opening
3	I	N	Opening
4	L	A	Opening
5	R	X	Opening
6	B	+	Opening
7	C	—	Opening
8	D	Y	Closing
9	E	Z	Closing
		P	Special closing
		#	Special
0 ≤ character ≤ 9	10 ≤ character ≤ 19	character ≥ 20	

The Normal Input Program itself can be found in the Appendix: in this chapter we intend only to discuss its structure and the mechanism of its actions. This will be done by explaining how instructions are constructed, and how words are assembled.

11.1 Construction of Instruction Part.

In 10.3 we have seen that the vertical ladder is entered at different points for each character. The opening symbols N, A, X go from the vertical ladder directly into the "construction of instructions" part. Here instructions are constructed in four parts, by making use of registers *10*, *11*, *12*, and *14*.

Register 14 deals with the opening and supplementary symbols;
Register 12 deals with the 1st address;
Register 11 deals with the 2nd address;
Register 10 deals with the parameter.

Register 13 always contains the *store instruction* during normal input, by means of which the instruction, when assembled, is stored.

11.1.1 Entry with N or X.

When entering with X or N, X000, i.e. the word with the most significant digit zero, is written into 14. Also, in the case of N, the address+1 from the store instruction in 13 is placed in 10.

11.1.2 Entry with A.

When entering with A, A000, i.e. the word with the most significant digit 1, is written into 14.

Subsequent supplementary symbols are added into appropriate positions in 14 from a supplementary symbols list; a decimal to binary conversion of addresses is made, these being temporarily stored in registers 11 and 12.

11.1.3 After Y.

When the word has been assembled (see 11.2) it is written into register 13. The program then jumps back to the reading of characters in the Short Input Program. Thus, after Y, blank tape can be written and this will be skipped over. Register 13 is normally the store instruction register; so the word before Y is usually of the form ADm, where m is the store instruction. The instruction ADmY can therefore be read as meaning "to begin input at m".

11.1.4 After Z.

After Z, the word just assembled is executed immediately. This enables an instruction during input to be executed very simply. For example, a program tape may begin :

	AD100Y	Begin input at 100 ;
	———	
	———	
	———	
and end with :	X000Z	Cease input and stop ;
or :	X100Z	Cease input and commence execution at 100.

A jump instruction can be executed during input by writing "jump....Y", for then the jump is written in 13 and subsequently executed. This is the mechanism used in the Short Input Program for executing a jump during input. The disadvantage is two-fold :

(1) Another instruction has to be read before the jump is executed, this other instruction being just thrown away.

(2) The store instruction is destroyed, so that all record of where the jump occurred has been lost.

For both these reasons "jump Z" is preferable. Z can also be attached to A instructions to be executed during input.

Example :

———	Normal input
———	
———	
AC100Z	(100) ⟶ cleared A
A200Z	(200) + (100) ⟶ A
AD300Z	(200) + (100) ⟶ (300)
———	Resume input
———	
———	

But the accumulators themselves are being used during the construction and assembly of the instruction AC100, etc. To overcome this, *register 4* is used as a *"phantom accumulator"* during execution of Z instructions.

The action of AC100Z is : assemble word AC100,
contents of 4 ⟶ A,
execute AC100 ; (100) ⟶ A.

In the case of jump....Z, this part is never reached. (A) ⟶ 4, Return to S.I.P.

Implicit in an A...Z instruction is the action (A) ⟶ 4. It is also worth noting that during the execution of a Z instruction (13) ⟶ B.

Thus, in the instruction ABD1000Z, the store instruction ⟶ 1000.

11.2 Word Assembly : Horizontal Ladder.

When a symbol > 20 is read the construction part is left and the *"horizontal ladder"* jumped to. This is another list of jump instructions which are written 'one' at the beginning of each track full of instructions which constitute the Normal Input Program. The Normal Input Program occupies 366 locations, i.e. 11½ tracks.

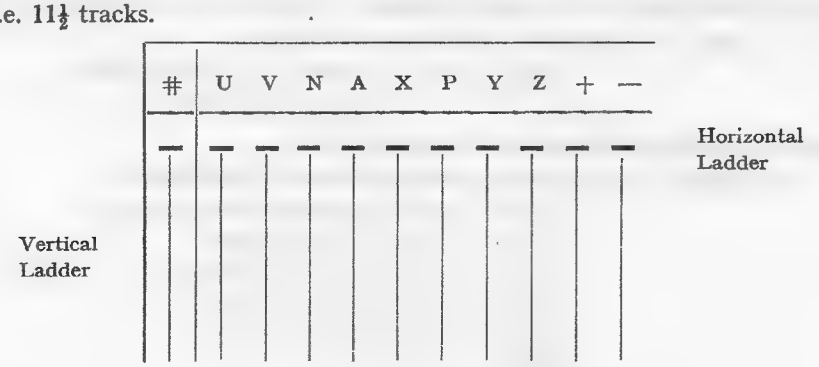


FIG. 11.3

The jumps corresponding to +, —, N, A, X, T, Z are all to the *word assembly*. The various parts of the word are assembled together, and four possibilities are present.

(1) Drum Address (D.A.) followed by Register Address (R.A.). This has taken place during construction :
D.A. in register 12, and
R.A. in register 11.

(2) Drum Address only : D.A. in register 11 ;
(12) is interpreted as R.A. zero.

(3) Register Address followed by Drum Address : R.A. in 12 ;
D.A. in 11.

(4) Register Address only : R.A. in register 11 ;
(12) interpreted as D.A. zero.

It is here that the conventions for Drum Address or Register Address being first are catered for.

After the assembly there is a special exit corresponding to each of the above entrances. The exits corresponding to +, —, N, A, X, T all cause the word just assembled to be stored according to the store instruction and then to jump back to the appropriate parts of the Normal Input Program ; for example, N, A, X go to the constructing of the next instruction, the storage instruction being increased by one.

11.3 The Special Symbols U and V.

When either U or V is read in the construction of instructions part, there is a jump to the horizontal ladder. In the case of U, there is a jump to read the next character, which is assumed to be an opening symbol or a digit $\leqslant 7$. (See 5.5.) In the former case, 0 is written in the V digit position, and there is a return to the appropriate entrance of normal construction. In the latter, 0 is written into the V digit position, and V4, V2, V1 filled in appropriately, and returns to normal construction to read the next character. When V is read the case is just the same except that now 1 is placed in the V digit position. Thus Um or Vm need not be at the end of the word, though it is usually better to put it there, i.e. A100BCV35 interpreted as A100BC5V3.

11.4 Parameter Facility : P.

It is possible to make an instruction relative to the contents of a register or drum location by simply writing an instruction of the form XaPb (see 7.5) which is interpreted as Xb + (a)—where b is always interpreted as the drum address, a may be the register address or drum address. For example, X5P100 = X100 + (5). With this P facility we can now write a program in an absolute form.

Example :

```
          AD9PY
9P0   |   N
9P1   |   NKKC
9P2   |   +1057
9P3   |   X9P5
9P4   |
→9P5  |
```

and this will be stored relative to a beginning point given in register 9.

11.4.1 Subroutine Call-in.

This gives a neat way of calling in a subroutine.

For example, the square-root subroutine is called in by the instruction X address KE4. In storing the square-root subroutine X address KE4 is written into location 50 with the appropriate address known at the time of storing. To call in the subroutine it is now only necessary to write X50P, because during input this will become

X000 + (50) = X000 + X address KE4
= X address KE4.

Hence, the correct call is written in automatically.

The "call-in" instructions for some common subroutines are :

X39P call in telephone dial (see 12.6).
X40P rounded multiplication.
X40P1 unrounded multiplication.
X41P single length division.
X42P double length division.
X50P square root.

11.4.2 Mechanism of the P Facility.

When P is read in the vertical ladder a jump to the horizontal ladder occurs, with the last address read in A. When entering the P part : AKQ000 $\longrightarrow$ 11 ; (10) + (address before P) $\longrightarrow$ 10. Normally (10) is initially zero.

Thus Xa a $\longrightarrow$ 11
XaP AKQ000 $\longrightarrow$ 11 ; (a) $\longrightarrow$ 10
XaPb b′ $\longrightarrow$ 11 ; (a) $\longrightarrow$ 10

Therefore XaPb = Xb′ + (a)

where the dash signifies that this address is regarded as a drum address. where a can be a register or drum address.

11.5 Cumulative Parameters.

There can be further P symbols in a single instruction ; but a second P is regarded differently. Here are some examples, showing the action.

XaPb : b′ $\longrightarrow$ 11 ; (a) $\longrightarrow$ 10 (′ signifies drum address).
XaPbP : AKQ000 $\longrightarrow$ 11 ; ((a) + b′) $\longrightarrow$ 10.
XaPbPc : c′ $\longrightarrow$ 11 ; ((a) + b′) $\longrightarrow$ 10.
Therefore XaPbPc = Xc′ + (b′ + (a)).
Similarly XaPbPcPd = Xd′ + (c′ + (b′ + (a))).

These are called *cumulative parameters.*

The Normal Input Program places a special interpretation on no address before P, provided this is a first P. It assumes that register 4 is meant. Thus XP30 = X030 + (4).

11.6 Accumulative Parameters.

If a point occurs after a P then a further P is regarded as a first P. Some examples, showing the action, are :

(1) XaPb : b′ $\longrightarrow$ 11 ; (a) $\longrightarrow$ 10.
XaPb.c : b′ $\longrightarrow$ 12 ; c $\longrightarrow$ 11 ; (a) $\longrightarrow$ 10.
Therefore XaPb.c = Xb′ + (a).c, where c is always regarded as a register address even if c > 31.

(2) XaPb.cP : AKQ000 $\longrightarrow$ 11 ; b′ $\longrightarrow$ 12 ; (a) + (c) $\longrightarrow$ 10. Here the P after the point is regarded as a first P.

(3) XaPb.cPd : d $\longrightarrow$ 11 ; b′ $\longrightarrow$ 12 ; (a) + (c) $\longrightarrow$ 10.
Therefore XaPb.cPd = Xb′ + (a) + (c).d, where d is regarded as a register address.

(4) Similarly XaP.bPc = X000 + (b) + (a).c
and XaP.bP.cPd = X000 + (c) + (b) + (a).d.

Rule : Last address = register address.
Last but one = drum address.
Previous ones are lost.

These are called *accumulative parameters.*

11.7 Some Parameter Conventions.

Register addresses 2 and 3 before a first P have a special meaning.

2P subtracts (12) from 10
3P subtracts ε from 10.

Parameters can also be attached to N instructions. In particular, the instruction N3P is an absolute way of writing a jump to the same address, but a register address must also be given even if this is zero.

Example : 200 | X200KE4U7 can be written in absolute form as N3PKE4U7.

It is worth mentioning here that a further drum address can be added to an N instruction, but a register address is also required to be given (even if it is zero) for it to be regarded as such.

Example : (1) 200 | N003.5 = X204.5
(2) 300 | N45.0 = X346

In this way any desired number of locations can be skipped over, whereas 200 | N003 = X201.3, i.e. 3 is regarded as a register address even though given as a drum address.

With the parameter facility we are able to write programs in a relative form without concerning ourselves where they are to be stored in the machine. Thus a tape can be headed with :

| A9CZ : (9) ⟶ A and then ⟶ 4
| ADPY : Begin to put in at (4)
P0
P1

THE NORMAL INPUT PROGRAM (2) *Chapter* 12

12.1 The T Facilities.

T is an opening symbol and is a character $\geqslant 20$ (see Fig. 11.2). When T is read in the Short Input Program there is a jump to the T part via the vertical ladder. T may also be read inside the construction of instructions part ; then the word before the T is assembled and stored ; thence a final jump to the T part.

12.1.1 T followed by Digit.

The digit m is regarded as a normal register address and the *address* from the store instruction is written into this register m. Thus m has only usual significance and is in the range $4 \leqslant m \leqslant 9$. It is possible to extend the facility to registers 10, 11, etc., by writing TK, TQ, and so on ; but since these registers are used during input this can only be done in exceptional circumstances.

T followed by a digit, i.e. Tm, enables registers 5, 6, 7, 8 to be used as special parameter registers during input.

This facility can also be used for writing a number into a register from the tape. For example, X49YT7 sends 49 $\longrightarrow$ 7. Tricks like this are of value in so-called *tape programs*. These are programs executed entirely from the tape and registers and are not written into the drum store.

Example :	X39PZ	Number from telephone
	AE9Z	dial $\longrightarrow$ 9

12.1.2 TA and TX Facility.

On reading A or X after T : A000 or X000 $\longrightarrow$ 14, and a further character is read which is normally a digit. This is again regarded as a register address m.

The action now is : store instruction address $\longrightarrow$ m ; previous m $\longrightarrow$ 10 ; and a return to the normal construction of instructions.

This gives :	r	TXm	:	r $\longrightarrow$ m X000 + (m) $\longrightarrow$ r
Similarly :	r	TAm	:	r $\longrightarrow$ m A000 + (m) $\longrightarrow$ r, where r is the store instruction address.

The purpose of this will become clear after a description of the TP facility.

12.1.3 TP Facility : Floating Addresses.

On reading P after T a jump back to the Short Input Program occurs ; P is read again and re-enters the *vertical ladder entrance for P*. This is P after T and must not be confused with parameter P, which is dealt with via the horizontal ladder. The next character, which is normally a digit, is now read, and is again regarded as a register address.

The Tm, TXm, and TPm instructions provide a *floating* address facility.

Example : Let us suppose that it is required to jump to an unknown point, i.e. drum location, and that further jumps are necessary to this location from other addresses.

r is the 1st point from which it is required to jump to an unknown point ;
r′ is the 2nd point, etc

r	TmX :	The presence of the instruction X means that the W digit is automatically inserted. r $\longrightarrow$ m ; XWO $\longrightarrow$ r.
–		
–		
r′	TXm :	X000 + (m) = Xr $\longrightarrow$ r′ ; r′ $\longrightarrow$ m.
–		
–		
r″	TXm :	X000 + (m) = Xr′ $\longrightarrow$ r″ ; r″ $\longrightarrow$ m.
–		
–		
r‴	TXm :	X000 + (m) = Xr″ $\longrightarrow$ r‴ ; r‴ $\longrightarrow$ m.
–		
–		
–		

Points to which jump is required = s I.e. after TPm all the jump instructions to this hitherto unknown point are made proper jump instructions Xs.	s – – – – – – – – – – – – z	TPm :	(m) ⟶ r‴ ; r‴ ⟶ r″. Replace Xr″ by Xs ⟶ r‴. Test for W digit—no W here, so repeat process. r″ ⟶ Xr′ ; replace Xr′ by Xs. No W digit here. r′ = Xr ; replace Xr by Xs. No W digit. r = XWO. W digit present ; therefore replace (m) by s. (i.e. s ⟶ m).
		XmP :	if required to jump back to s.

Although there is no limit to the number of jumps that can be made, there is a certain restriction in that m is only in the range $4 \leqslant m \leqslant 9$ and there are, therefore, only six possible floating points.

With the *TV facility*, however, ultimate flexibility in floating address can be achieved.

12.1.4 TV Facility.

This facility gives an extended facility to that described in 12.1.3. The TV facility works in conjunction with the Normal Input Program but has to be entered as a separate program. The ~~latter~~ former suffers two limitations :

(i) m is limited to the range $4 \leqslant m \leqslant 9$;

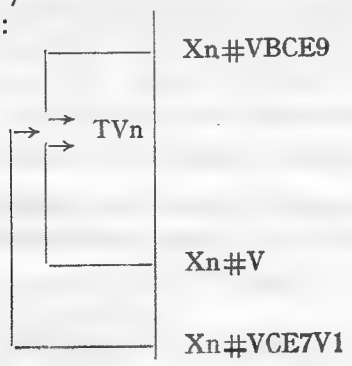
(ii) it must be known whether the instruction containing the floating address is entered into the computor before or after the labelled point, i.e. before or after in time.

In the TV facility both these limitations are removed. The rules are :

(1) An unknown drum address in an instruction is written as n#V where n is any drum location set aside for the purpose. n must be cleared before use.

(2) The unknown location when entered is labelled TVn with the same n.

Thus :



It is envisaged that the TV facility is best used when it is required to use floating addresses between complete programs. The floating address facility included in the Normal Input Program, and described in 12.1.3, is best used inside a particular program.

12.1.5 TE Facility.

When E is read after T there is a return to the Short Input Program and thence to the E entrance of the vertical ladder.

The TE facility enables the difference of two instructions to be obtained during input. For example, it is required to input X200BE4 — X102K3. To achieve this during input we simply attach TE to the instructions, i.e. X200BE4 — X102K3TE. In this way the correct difference is formed and written into the store.

This can be extended to form an alternating sum of instructions by placing a further E on the instructions, i.e.

$$X_1 - X_2 \quad TE$$
$$X_1 - X_2 + X_3 \quad TEE$$
$$X_1 - X_2 + X_3 - X_4 \quad TEEE$$

where X_n denotes any instruction.

It is important to know how the machine reacts to an instruction of this nature. In the instruction $X_1 - X_2$ TE, the machine firstly reads X_1 (which is really a complete instruction) and stores it in, say, location n. The sign — is stored in $n + 1$, and X_2 in $n + 2$. Then TE is read, and the binary result is calculated and placed in n. Subsequent instructions will then destroy $(n + 1)$ and $(n + 2)$. In the same way $X_1 - X_2 + X_3 - X_4$ will originally take up seven locations before TEEE is read and the calculated result placed in n.

This must be remembered when it is required to insert a TE instruction into a program which is already stored. In this event, the original contents of the appropriate locations which follow must be restored.

12.1.6 **TD Facility.**

After Tm the machine returns to the Short Input Program. If D is now read, the *directory part* of the Normal Input Program is entered via the vertical ladder. The directory part only works if the tape has begun with the *standard beginning with directory*. The standard beginning is :

	AD100YT4	:	Begin input at 100 if U7 = 0.
	X39PU7Z	:	U7 = 1 (i.e. start key not depressed) ; begin input at dialled location (see 12.5).
	AD34Z	:	Key address ⟶ location 34.
	ADPY	:	Begin to put in at P0.
P0	NDK11Q	:	1st store instruction of directory.
	ADP16Y	:	Keep 16 places free.
	T9D	:	Use directory ; place beginning of next program in 9 and in P1 of directory.

Suppose we wish to store a main program and three subroutines. This is placed on the tape as :

—Standard
—beginning
—
—
Blank may follow
—
— 1st subroutine : ends with
— T9D
—
Blank
—
— 2nd subroutine : ends with
— T9D
—
Blank
—
— 3rd subroutine : ends with
— T9D
—
Blank
—
— Main
— program
—
Blank
—
— Standard
— ending
—

Input begins at the dialled beginning if the start key is not depressed, i.e. it waits for the dial. It begins at 100 otherwise. The standard beginning places the working instruction in the beginning point P0, and P0 ⟶ 34. This makes the storing instruction begin input at P16.

T9D, which ends the standard beginning, sends P16 ⟶ P1 and P16 ⟶ 9. The actual input of the 1st subroutine commences at P16. A note of this address is now in P1. Similarly, the 1st subroutine ends with T9D. This puts the beginning point of the 2nd subroutine ⟶ P2 and ⟶ 9. Thus, the beginning point of each subroutine is written temporarily in 9 and permanently in P1, P2, P3. The end of the last subroutine places the beginning of the main program in P4 (in this special case of three subroutines), and the ending +1 is placed in P5 by the *standard ending*. Also, the beginning of the main program is finally written into P0.

Therefore, the directory for three subroutines consists of :

P0	Beginning address of main program.
P1	Beginning address of 1st subroutine.
P2	Beginning address of 2nd subroutine.
P3	Beginning address of 3rd subroutine.
P4	Beginning address of main program.
P5	Ending address + 1 of main program.

The standard ending is :

T0D	:	Use directory to record last address of main program.
AC9Z	:	Call in (9).
AD34PZ	:	Place beginning of main program in the key address.
XZ	:	Stops tape. This must be X34Z if the directory is to be used on unattended work.

12.2 Input of Numbers.

The Normal Input Program has an "input of numbers" part, and can distinguish between + and — when these symbols are read from the tape. A point following either symbol is also remembered. The following digits are converted to binary as if the number were a positive integer. Positive integers are represented as $X2^{-32}$, with their least significant digit in the least significant digit position of the word.

If there was a point, there is a final correction made to the conversion to achieve the correct conversion of the fraction. In fact, the conversion of fraction f has been $f \times 10^9 \times 2^{-32}$. We must now correct this by multiplying by $10^{-9} \times 2^{32}$. (See 3.2.) Finally, the word is made negative if the sign — was present. All this is done after the reading of a new opening symbol. When the opening symbol of a new word appears, these corrections take place and the number is stored. After storing, the machine returns to the Short Input Program. Thus, in reading a series of numbers, re-entry to the "input of numbers" part is always via the Short Input Program and the vertical ladder.

Therefore, during input of numerical data, the machine is always prepared to begin input in any code.

12.3 Correction Facility: #.

The erase or correction symbol, which consists of all the five holes on the tape, has a special written symbol #. If a word is mispunched during tape preparation, we have only to punch # and begin the word again, which may be an instruction or a number.

The action of # causes the machine to jump back to the Short Input Program without storing the partially built-up word. The jump instruction to the Short Input Program associated with # is in the position of the Normal Input Program shared by the horizontal and vertical ladders (see Fig. 11.1). Thus we may place a number of consecutive #'s on the tape.

12.4 Subroutine for Taking in Numbers: X33P.

The subroutine is called in by X33P, and makes use of the input of numbers part of the Normal Input Program. Since the N.I.P. uses registers 11, 12, and 13, these are "brought to safety" in the drum before the number is input and then replaced afterwards.

12.5 Subroutine Conventions.

The *subroutine convention* is that registers 10–14 inclusive must be left undisturbed by the subroutine. A subroutine user can leave results in 10–14 during the action of the subroutine, but if he leaves results in 4–9 inclusive or in 15 they may be spoilt. To find out if a particular subroutine does not use these registers, it is necessary to look up the particular subroutine specification.

When subroutines and main program are stored with the directory, it is possible to refer in the main program to each subroutine as the 2nd or 3rd, etc.

Example:

```
—
—
X34P2PP  :  Jump to 2nd subroutine.
—
—
—
```

This is because X34PkPP is the 1st instruction of the subroutine k, i.e.:

Because	(34)	=	P0.
Therefore	34P	=	P0 of directory.
	34Pk	=	Pk of directory.
	34PkP	=	(Pk) of directory = beginning address of kth subroutine.
	34PkPP	=	(beginning address of kth subroutine), i.e. 1st instruction of kth subroutine.

Example: The square-root subroutine begins:

```
          AC9Z     :  Take the beginning point from 9.
          AD50Y    :
50   |    XP1KE8   :  Call in instruction 50.
          ADPY
P0   |    NKE8
P1   |    —        :  P1 is the actual beginning.
     |    —
     |    —
     |    —
     |    T9D
```

Then the square root can be called in either by X50P or by X34PkPP, when it is the kth subroutine in a directory.

The Normal Input Program also contains within itself the *multiplication subroutine*, called in by X40P (rounded) and X40P1 (unrounded), and a *telephone dial subroutine* called in by X39P.

12.6 The Telephone Dial Program: X39P.

Entered by X39P, this causes the machine to stop and wait for a number to be dialled. An integer is dialled into the machine in decimal form, digit by digit. The number is automatically converted into binary, the binary integer being built up in A. The machine can cope with integers of up to nine decimal digits. The end of a number is assumed by the machine if more than five seconds elapse after a digit is dialled.

Then, the number ⟶ A, and the number of decimal digits ⟶ B; the machine then returns to executing the next instruction in the main program.

A misdialled number can be cleared during its input by depressing the start key and beginning to dial again.

OUTPUT PROGRAMS

Output programs fall into two categories :

(1) Complete and comprehensive output programs.
(2) Building blocks.

13.1 Complete Output Program.

The value of the complete output program is the ease with which it can be used. It is possible to :

(1) Print.
(2) Print and punch tape in machine code simultaneously.
(3) Punch in machine code.
(4) Punch in teleprinter code.

The layout of printed information can be specified by supplying a *single pattern word.*

There are two versions of the output program. One is timed to run the 25 character per second (25 c/s) punch, the other the 50 c/s punch. The programmer places the number to be output in A and, if specifying the layout, the pattern in B.

The output program, which is given in detail in the Appendix, is called in like a subroutine by an instruction of the form XmPn. m can have various values with different meanings :

m = 45 : Print.
m = 46 : Print and punch tape in machine code simultaneously.
m = 47 : Punch in teleprinter code.
m = 48 : Punch in machine code (always has n = 0).
m = 49 : Standard program for punching in short or *binary code* (always has n = 0).

Similarly, n has different values :

n = 0 : Output according to *resident pattern*, i.e. that which has been already set in the output program.
n = 2 : Output according to *incident pattern*, i.e. the one placed in B.
n = 28 : Output *superpositive** integer according to resident pattern, i.e. sign digit being treated as part of the number.
n = 30 : Output superpositive integer according to incident pattern.
n = 31 : Output integer in normal form, i.e. ± XXXXX space XXXXX space space.
n = 32 : Output fraction in normal form, i.e. $\pm \overset{1}{0}$.XXXXXXXXX space space.
n = 4C : Carriage return, line feed, figure shift.

* A superpositive integer is a +ve integer whose sign digit is regarded as the most significant digit of the integer.

Some instructions of this form have special meanings :

(1) X47P3 : Make resident pattern = (B).
(2) X45P6, X47P6 : Output thirteen pentads* from A and B. Output ceases when (A) = 0. The thirteen pentads do not include the a_0 digit.
When called in by X45P6C or X47P6C, the **B** accumulator is cleared and only six pentads are output from A. This output is useful for text, etc.

* A pentad is a group of five digits and is made up in the reverse order to that in which the character is normally written. E.g., the letter L is represented by 01101, but is in the machine as 10110.

13.1.1 Layout Pattern.

This is a 33-binary digit word in which digits, or groups of digits, are used to specify the output layout. Each digit has a specific meaning.

P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	....	P28	P29	P30	P31	P32
X	X	X	X	X	X	X	X	X	X	X	X		X	X	X	X	X

P0 = 1 : Type or punch in teleprinter code the sign of the number.
P0 = 0 : Suppress the sign.

In machine code output of the sign is always provided regardless of P0.

P1 = 1 : Number to be converted and output as a fraction.

P1 = 0 : Number to be converted and output as an integer.

In machine code the point is punched automatically. In other forms this depends on the pattern.

P2, P3, P4, P5 : together specify the number of digits to be suppressed.

If there is no suppression, i.e. when P2, P3, P4, P5 = 0, then 10 digits are output. The digits specified for output are the most significant in the number.

This can be written as :

If P2, P3, P4, P5 = i, then output = 10 — i digits. So when the combination is : P2 P3 P4 P5, for example, five digits are output and five digits suppressed.
0 1 0 1

The following digits are taken in pairs, e.g. P6 and P7, P8 and P9, etc. The value of such a "bit pair" is called k, where $0 \leqslant k \leqslant 3$.

k = 0 : Output next digit *imperatively*, i.e. the next digit must be printed whatever it may be.

k = 1 : Output next digit *facultatively*, or conditionally. Facultative output suppresses the most significant zeros replacing them by spaces. When a non-zero digit is found, subsequent output becomes imperative.

k = 2 : Type or punch in teleprinter code a space.

k = 3 : Type or punch in teleprinter code a point. After output of a point, output of digits becomes imperative.

When all the 10 — i digits have been output, the values k = 0 and k = 1 change in meaning :

k = 0 : Leave output program and return to main program.

k = 1 : Carriage return, line feed, figure shift.

The last digit P32 is automatically followed by a zero :

P32 = 0 : Leave output program.

P32 = 1 : Type space ; after which k = 0 follows.

In fraction output the most significant digit is regarded as the digit *in front of the point.* This may be 0 or 1 as a result of rounding, e.g. 0.1111 ..... 1 will be output as 1.00 ..... 0. For fraction output P6, P7 = 2 has a special meaning : suppress this most significant digit. This is useful for double length output where it is wished to output the tail next to the head, with the digit in front of the point in the tail suppressed.

The slow version of the output program reconverts the number converted for output and leaves it in A and B. This can be used for checking purposes.

13.1.2 Some Samples of Output Pattern.

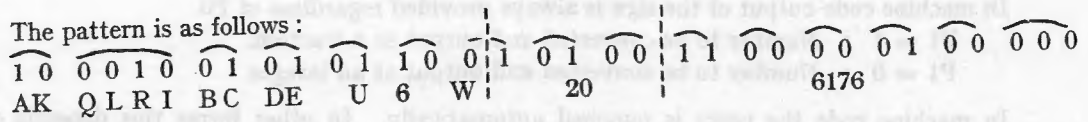
Here are some samples of output patterns :

Integer ± XXX spa XXX.XX
± XXX spa XXX.XX
± XXX spa XXX.XX
etc., with only the one digit in front of the point imperative.

It is possible to output a column like the following example :

```
+ 123 498.17
— 23 001.01
+     107.30
—      23.69
+       0.98
```

The pattern is as follows :

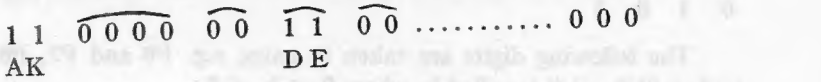


The pattern can be expressed as a "pseudo-instruction", by writing letters corresponding to ones in the function part, and by converting the binary register address part into their decimal equivalent.

The pseudo-instruction in this case is A6176RCE20U6.

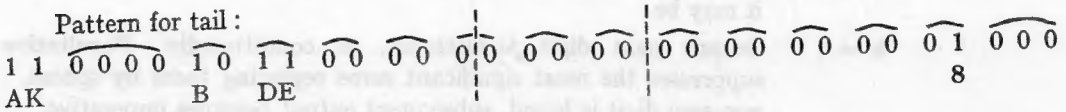
13.1.3 Signed Double Length Fractions.

The patterns for the head and tail of double length fractions with sign are :



The pseudo-instruction is A000KDE.

Pattern for tail :



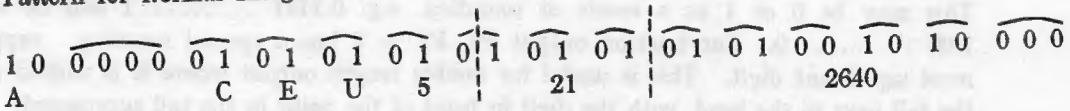
The pseudo-instruction is A008KBDE.

Care must be taken that the tail does not round up to 1 on output. This can be done by ensuring that the last three digits of the tail are zeros. There is then a slight doubt on the accuracy of the least significant digit of the tail.

When an integer contains more digits than has been allowed for by the pattern, the machine stops. On restarting :

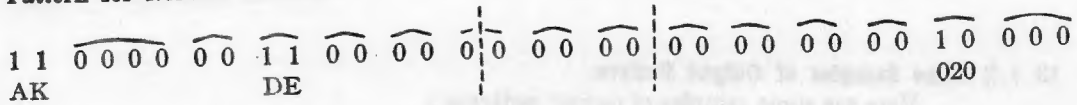
(1) The slow program leaves the space blank.
(2) The fast program writes zero in the space.

13.1.4 Pattern for Normal Integer.



Pseudo-instruction is A2640CE21U5.

13.1.5 Pattern for Normal Fraction.



Pseudo-instruction is A020KDE.

13.2 Building Blocks.

Building blocks consist of short pieces of program, each designed to give a particular facility. Typical blocks are :

Output digit block ;
Output sign block ;
Output carriage return, line feed, figure shift.

The advantage of the building block idea is that only those blocks which are required for a specific purpose need be used. One can achieve as much complexity of output as required, but will fill no more storage space than necessary.

13.2.1 Building Block Program for Output of Digit and Sign.

P0	NLBE6
P1	ACE25
"P2	+0
P3	NLB6
P4	NLBV
P5	NKKCE6R
"P6	X009–X000K6RTE
P7	NK6R
P8	NKKK2C
P9	f(0)
P10	f(1) Table of teleprinter
P11	f(2) equivalents of the
P12	f(3) digits 0–9.
P13	f(4)
P14	f(5) For example :
P15	f(6) f(6) = 0.11 00 11 00 11 11 etc.
P16	f(7) start 1 0 1 0 1 stop
P17	f(8) These constants convert machine
P18	f(9) code into teleprinter code.
P19	NKE6
P20	X5K13
P21	XP31K4
—	
—	
—	
P31	—1

Program to Set Sign.

P32	NIBC3
P33	NKKCE25
"P34	AP2D25
P35	NKKCE5
"P36	f(—)
P37	NKKCK1V2
"P38	f(+)
P39	NIBC3
P40	XP19

The sign program precedes the typing out of digits, i.e. on first entering the machine the instruction XP32KE4C enters the subroutine at instruction P32 and the sign is printed out. For this reason, let us firstly analyse the sign program. Since B contains the number to be printed out, we will treat the digit in B as (1) positive, (2) negative.

A	B	C	D	Remarks
	(1) +ve	XP33IBC3		
	—ve	AP34CE25	XP35IBC3	Sets teleprinter signal ; does not double (A).
AP2D25				
	—ve	XP35IBC3		
	+ve	AP36CE5	XP37IBC3	AP2D25 ⟶ 5
f(—)	+ve	XP37IBC3		
f(—)	—ve	AP38KC1V2	XP39IBC3	Clears A ; test succeeds on sign of B. f(+) ⟶ A. Note action of K1.
f(+)	—ve	XP40IBC3		
f(+)	+ve	XP19		

Here the test succeeds and XP19 jumps to the instruction XP20KE6 in P19. Let us now consider the case of a negative number in B. In this case the test fails.

f(—)	(2) —ve	XP37IBC3		
f(—)	+ve	AP38KC1V2	XP39IBC3	Test fails. So next instruction is not executed.
f(—)	+ve			
f(—)	+ve	XP40IBC3		
f(—)	—ve	XP19		

Now the sign is printed out :

A	B	C	D	Remarks
		XP19 XP20KE6 X5K13 ADP2.25 X5K12 until X6K XP21 XP31K4	XP21 X5K12 X6K	(D) = XP21 ⟶ 6 = return instruction.

The repeat instruction X5K13 repeats ADP2.25 thirteen times. This sets the teleprinter signal and doubles (A), and is executed once per 10 ms. owing to the harmless writing into P2. By the repeated execution of AP2D25 the teleprinter signal is each time made equal to a_0 ; also, A is left-shifted because of the action of register 25, so that all consecutive digits come into the a_0 position. The process causes each signal to last 20 ms., i.e. it duplicates each digit.

Successive digits are printed by entering with XP0KE4C. The contents of five are made AP2D25 by the Sign Program.

Instruction Analysis :

XP1LBE6 : Store (B) in 6 ; form 2 (B).

ACE25 : Clear A ; set teleprinter signal low (do not double A) ; sends (P2) = +0 ⟶ A.
XP3LBE6 returns to C from D ; sends 2 (B) ⟶ 6. Forms 4 (B).

XP4LB6 : Forms 8 (B) + 2 (B) = 10 (B) = B′.

XP5LBV : Forms 2 (B)′. The carry trap is set to zero, i.e. V releases carry trap, previously set by the sum 8 (B) + 2(B). Because there is no register address, nothing is added into B, and hence the carry trap is set to zero.

AP6CE6R : Store digit to be typed in 6 ; the constant in P6 = X009–X000K6R ⟶ cleared A.
XP5LBV returns to C as XP7LBV. Forms 2(B)′ and 2(P6).

XP8K6R : Forms (B)′ and (P6) in A ; passes into D to become XP10K6R.

AP9K2C : K2 connects C to register 2 = (A) ; executes X009–X000K6R + XP10K6R (from D) = XP19.

XP20KE6 : Write return instruction ⟶ 6.

Successive digits are printed out by entering with XP0KE4C, and they are output in the same way as the sign, i.e. by means of the instruction AP2D25, which is repeated thirteen times.